

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

July 18, 2025

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

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

8 \msg_new:nnn { nicematrix } { latex-too-old }
9 {
10   Your-LaTeX-release-is-too-old. \
11   You-need-at-least-a-the-version-of-2023-11-01
12 }

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

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

*This document corresponds to the version 7.1e of `nicematrix`, at the date of 2025/07/18.

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

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

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

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

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

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

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

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

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

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

We try to detect whether the compilation is done on Overleaf. We use `\c_sys_jobname_str` because, with Overleaf, the value of `\c_sys_jobname_str` is always “output”.

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

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

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

2 Collecting options

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

Example :

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

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

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

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

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

```

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

```

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

```

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

```

3 Technical definitions

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

```

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

```

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

```

89 \tl_new:N \l_@@_argspec_tl

```

```

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

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

```

In some constructions, we will have to use a `{pgfpicture}` which *must* be replaced by a `{tikzpicture}` if Tikz is loaded. However, this switch between `{pgfpicture}` and `{tikzpicture}` can't be done dynamically with a conditional because, when the Tikz library `external` is loaded by the user, the pair `\tikzpicture-\endtikzpicture` (or `\begin{tikzpicture}-\end{tikzpicture}`) must be statically “visible” (even when externalization is not activated).

That's why we create `\c_@@_pgfortikzpicture_tl` and `\c_@@_endpgfortikzpicture_tl` which will be used to construct in a `\hook_gput_code:nnn { begindocument } { . }` the correct version of some commands. The tokens `\exp_not:N` are mandatory.

```

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

```

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

```

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

```

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

```

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

```

If the final user uses `nicematrix`, PGF/Tikz will write instruction `\pgfsyspdfmark` in the `aux` file. If he changes its mind and no longer loads `nicematrix`, an error may occur at the next compilation because of remanent instructions `\pgfsyspdfmark` in the `aux` file. With the following code, we try to avoid that situation.

```

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

```

We define a command `\iddots` similar to `\ddots` (``) but with dots going forward (``). We use `\ProvideDocumentCommand` and so, if the command `\iddots` has already been defined (for example by the package `mathdots`), we don't define it again.

```

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

```

This definition is a variant of the standard definition of `\ddots`.

In the `aux` file, we will have the references of the PGF/Tikz nodes created by `nicematrix`. However, when `booktabs` is used, some nodes (more precisely, some `row` nodes) will be defined twice because their position will be modified. In order to avoid an error message in this case, we will redefine `\pgfutil@check@rerun` in the `aux` file.

```

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

```

The new version of `\pgfutil@check@rerun` will not check the PGF nodes whose names start with `nm-` (which is the prefix for the nodes created by `nicematrix`).

```

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

```

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

```

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

```

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

```

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

```

```

177     }
178 }

```

When `colortbl` is used, we have to catch the tokens `\columncolor` in the preamble because, otherwise, `colortbl` will catch them and the colored panels won't be drawn by `nicematrix` but by `colortbl` (with an output which is not perfect).

```

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

```

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

```

184     }
185 }
186 {
187     \cs_new_protected:Npn \@@_revert_colortbl: { }
188     \cs_new_protected:Npn \@@_replace_columncolor:
189     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }

```

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

```

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

```

Idem for `\CT@drs@`.

```

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

```

We have to redefine `\cline` for several reasons. The command `\@@_cline:` will be linked to `\cline` in the beginning of `{NiceArrayWithDelims}`. The following commands must *not* be protected.

```

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

```

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

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

Our `\everycr` has been modified. In particular, the creation of the `row` node is in the `\everycr` (maybe we should put it with the incrementation of `\c@iRow`). Since the following `\cr` correspond to a “false row”, we have to nullify `\everycr`.

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

The following version of `\cline` spreads the array of a quantity equal to `\arrayrulewidth` as does `\hline`. It will be loaded excepted if the key `standard-cline` has been used.

```
231 \cs_set:Npn \@@_cline:
```

We have to act in a fully expandable way since there may be `\noalign` (in the `\multispan`) to detect. That’s why we use `\@@_cline_i:en`.

```
232 { \@@_cline_i:en \l_@@_first_col_int }
```

The command `\cline_i:nn` has two arguments. The first is the number of the current column (it *must* be used in that column). The second is a standard argument of `\cline` of the form *i-j* or the form *i*.

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

Now, `#1` is the number of the current column and we have to draw a line from the column `#2` to the column `#3` (both included).

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

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

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

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

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

¹See question 99041 on TeX StackExchange.

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

295 \dim_new:N \l_@@_tmpc_dim
296 \dim_new:N \l_@@_tmpd_dim
297 \dim_new:N \l_@@_tmpe_dim
298 \dim_new:N \l_@@_tmpf_dim

299 \tl_new:N \l_@@_tmpc_tl
300 \tl_new:N \l_@@_tmpd_tl

301 \int_new:N \l_@@_tmpc_int

```


4 Parameters

The following counter will count the environments `{NiceArray}`. The value of this counter will be used to prefix the names of the Tikz nodes created in the array.

```
302 \int_new:N \g_@@_env_int
```

The following command is only a syntactic shortcut. It must *not* be protected (it will be used in names of PGF nodes).

```
303 \cs_new:Npn \@@_env: { nm - \int_use:N \g_@@_env_int }
```

The command `\NiceMatrixLastEnv` is not used by the package `nicematrix`. It's only a facility given to the final user. It gives the number of the last environment (in fact the number of the current environment but it's meant to be used after the environment in order to refer to that environment — and its nodes — without having to give it a name). This command *must* be expandable since it will be used in pgf nodes.

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

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

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

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

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

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

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

In fact, if there is delimiters in the preamble of `{NiceArray}` (eg: `[cccc]`), this boolean will be set to false.

The following boolean will be equal to `true` in the environments which have a preamble (provided by the final user): `{NiceTabular}`, `{NiceArray}`, `{pNiceArray}`, etc.

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

We need a special treatment for `{NiceMatrix}` when `vlines` is not used, in order to retrieve `\arraycolsep` on both sides.

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

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

```
314 \int_new:N \g_@@_NiceMatrixBlock_int
```

It's possible to put tabular notes (with `\tabularnote`) in the caption if that caption is composed *above* the tabular. In such case, we will count in `\g_@@_notes_caption_int` the number of uses of the command `\tabularnote` *without optional argument* in that caption.

```
315 \int_new:N \g_@@_notes_caption_int
```

The dimension `\l_@@_columns_width_dim` will be used when the options specify that all the columns must have the same width (but, if the key `columns-width` is used with the special value `auto`, the boolean `\l_@@_auto_columns_width_bool` also will be raised).

```
316 \dim_new:N \l_@@_columns_width_dim
```

The dimension `\l_@@_col_width_dim` will be available in each cell which belongs to a column of fixed width: `w{...}{...}`, `W{...}{...}`, `p{...}`, `m{...}`, `b{...}` but also `X` (when the actual width of that column is known, that is to say after the first compilation). It's the width of that column. It will be used by some commands `\Block`. A non positive value means that the column has no fixed width (it's a column of type `c`, `r`, `l`, etc.).

```
317 \dim_new:N \l_@@_col_width_dim
318 \dim_set:Nn \l_@@_col_width_dim { -1 cm }
```

The following counters will be used to count the numbers of rows and columns of the array.

```
319 \int_new:N \g_@@_row_total_int
320 \int_new:N \g_@@_col_total_int
```

The following parameter will be used by `\@@_create_row_node`: to avoid to create the same row-node twice (at the end of the array).

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

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

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

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

```
323 \tl_new:N \l_@@_hpos_cell_tl
324 \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
```

When there is a mono-column block (created by the command `\Block`), we want to take into account the width of that block for the width of the column. That's why we compute the width of that block in the `\g_@@_blocks_wd_dim` and, after the construction of the box `\l_@@_cell_box`, we change the width of that box to take into account the length `\g_@@_blocks_wd_dim`.

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

Idem for the mono-row blocks.

```
326 \dim_new:N \g_@@_blocks_ht_dim
327 \dim_new:N \g_@@_blocks_dp_dim
```

The following dimension correspond to the key `width` (which may be fixed in `\NiceMatrixOptions` but also in an environment `{NiceTabular}`).

```
328 \dim_new:N \l_@@_width_dim
```

The clist `\g_@@_names_clist` will be the list of all the names of environments used (via the option `name`) in the document: two environments must not have the same name. However, it's possible to use the option `allow-duplicate-names`.

```
329 \clist_new:N \g_@@_names_clist
```

We want to know whether we are in an environment of `nicematrix` because we will raise an error if the user tries to use nested environments.

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

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

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

```
333 \bool_new:N \l_@@_initial_open_bool
334 \bool_new:N \l_@@_final_open_bool
335 \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.

```
336 \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).

```
337 \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).

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

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

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

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

```
340 \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.

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

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

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

```
343 \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 }`).

```
344 \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.

```
345 \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.

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

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

```
348 \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}`).

```
349 \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`).

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

For `\multicolumn`.

```
351 \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`.

```
352 \tl_new:N \l_@@_columns_type_tl
353 \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 `:`.

```
354 \tl_new:N \l_@@_xdots_down_tl
355 \tl_new:N \l_@@_xdots_up_tl
356 \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`).

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

358 \cs_new_protected:Npn \@@_test_if_math_mode:
359 {
360   \if_mode_math: \else:
361     \@@_fatal:n { Outside-math-mode }
362   \fi:
363 }
```

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

```
364 \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”.

```
365 \colorlet { nicematrix-last-col } { . }
366 \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`).

```
367 \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*.

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

```
370 \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)`.

```
371 \cs_new:Npn \@@_full_name_env:
372 {
373   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
374     { command \space \c_backslash_str \g_@@_name_env_str }
375     { environment \space \{ \g_@@_name_env_str \} }
376 }
```

```
377 \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.

```
378 \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.

```
379 \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.

```
380 \tl_new:N \g_@@_pre_code_before_tl
381 \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.

```
382 \tl_new:N \g_@@_pre_code_after_tl
383 \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.

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

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

```
385 \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.

```
386 \int_new:N \l_@@_old_iRow_int
387 \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).

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

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

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

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

```
390 \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.

```
391 \bool_new:N \l_@@_X_columns_aux_bool
392 \dim_new:N \l_@@_X_columns_dim
```

This boolean will be used only to detect in an expandable way whether we are at the beginning of the (potential) column zero, in order to raise an error if `\Hdotsfor` is used in that column.

```
393 \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).

```
394 \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).

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

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

```
397 \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).

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

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

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

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

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

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

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

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

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

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

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

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

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

```
409 \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”.

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

```
411 \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.

```
412 \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!).

```
413 \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.

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

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

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

```
415 \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.

```
416 \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`).

```
417 \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`).

```
418 \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.

```
419 \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).

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

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

By default, the diagonal lines will be parallelized². 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.

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

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

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

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the Δ_x and Δ_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 Δ_x and Δ_y of the first `\Iddots` diagonal.

```

424 \dim_new:N \g_@@_delta_x_one_dim
425 \dim_new:N \g_@@_delta_y_one_dim
426 \dim_new:N \g_@@_delta_x_two_dim
427 \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`).

```

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

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

```

The following counters will be used when drawing the rules.

```

436 \int_new:N \l_@@_start_int
437 \int_set_eq:NN \l_@@_start_int \c_one_int
438 \int_new:N \l_@@_end_int
439 \int_new:N \l_@@_local_start_int
440 \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.

```

441 \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).

```

442 \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`.

```

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

```

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

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

```

449 \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`.

```

450 \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.

```
451 \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.

```
452 \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`).

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

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

```
457 \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).

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

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

```
459 \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).

```
460 \bool_new:N \l_@@_vlines_block_bool
461 \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.

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

463 \dim_new:N \l_@@_submatrix_extra_height_dim
464 \dim_new:N \l_@@_submatrix_left_xshift_dim
465 \dim_new:N \l_@@_submatrix_right_xshift_dim
466 \clist_new:N \l_@@_hlines_clist
467 \clist_new:N \l_@@_vlines_clist
468 \clist_new:N \l_@@_submatrix_hlines_clist
469 \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}`).

```
470 \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.

```
471 \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`).

```
472 \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.

```
473 \int_new:N \l_@@_first_row_int
474 \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.

```
475 \int_new:N \l_@@_first_col_int
476 \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).

```
477 \int_new:N \l_@@_last_row_int
478 \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”.³

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

Idem for `\l_@@_last_col_without_value_bool`

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

- **Last column**

For the potential “last column”, we use an integer. A value of `-2` means that there is no last column. A value of `-1` means that we are in an environment without preamble (e.g. `{bNiceMatrix}`) and there is a last column but we don’t know its value because the user has used the option `last-col` without value. A value of 0 means that the option `last-col` has been used in an environment with preamble (like `{pNiceArray}`): in this case, the key was necessary without argument. The command `\NiceMatrixOptions` also sets `\l_@@_last_col_int` to 0.

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

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

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

```

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

```

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

```

483 \bool_new:N \g_@@_last_col_found_bool

```

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

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

```

484 \bool_new:N \l_@@_in_last_col_bool

```

Some utilities

```

485 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
486 {

```

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

```

487 \def \l_tmpa_tl { #1 }
488 \def \l_tmpb_tl { #2 }
489 }

```

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.

```

490 \cs_new_protected:Npn \@@_expand_clist:N #1
491 {
492 \clist_if_in:NnF #1 { all }
493 {
494 \clist_clear:N \l_tmpa_clist
495 \clist_map_inline:Nn #1
496 {

```

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

```

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

```

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

```

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

```

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.

```

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

```

5 The command `\tabularnote`

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

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

The LaTeX counter `tabularnote` will be used to count the tabular notes during the construction of the array (this counter won't be used during the composition of the notes at the end of the array). You use a LaTeX counter because we will use `\refstepcounter` in order to have the tabular notes referenceable.

```
514 \newcounter { tabularnote }
```

We want to avoid error messages for duplicate labels when the package `hyperref` is used. That's why we will count all the tabular notes of the whole document with `\g_@@_tabularnote_int`.

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

Before the actual tabular notes, it's possible to put a text specified by the key `tabularnote` of the environment. The token list `\g_@@_tabularnote_tl` corresponds to the value of that key.

```
519 \tl_new:N \g_@@_tabularnote_tl
```

We prepare the tools for the formatting of the references of the footnotes (in the tabular itself). There may have several references of footnote at the same point and we have to take into account that point.

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

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

```

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

```

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

```

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

```

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

```

528 \cs_new:Npn \@@_notes_label_in_tabular:n #1 { \textsuperscript { #1 } }

```

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

```

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

```

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

```

530 \cs_set:Npn \thetabularnote { { \@@_notes_style:n { tabularnote } } }

```

The tabular notes will be available for the final user only when `enumitem` is loaded. Indeed, the tabular notes will be composed at the end of the array with a list customized by `enumitem` (a list `tabularnotes` in the general case and a list `tabularnotes*` if the key `para` is in force). However, we can test whether `enumitem` has been loaded only at the beginning of the document (we want to allow the user to load `enumitem` after `nicematrix`).

```

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

```

The type of list `tabularnotes` will be used to format the tabular notes at the end of the array in the general case and `tabularnotes*` will be used if the key `para` is in force.

```

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

```

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

```

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

```

```

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

```

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

```

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

```

You have to see whether the argument of `\tabularnote` has yet been used as argument of another `\tabularnote` in the same tabular. In that case, there will be only one note (for both commands `\tabularnote`) at the end of the tabular. We search the argument of our command `\tabularnote` in `\g_@@_notes_seq`. The position in the sequence will be stored in `\l_tmpa_int` (0 if the text is not in the sequence yet).

```

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

```

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

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

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

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

```

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

```

```

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

```

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

```

624     \hbox_set:Nn \l_tmpa_box
625     {

```

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

```

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

```

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

```

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

```

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

```

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

```

If the command `\tabularnote` is used exactly at the end of the cell, the `\unskip` (inserted by `array`?) will delete the skip we insert now and the label of the footnote will be composed in an overlapping position (by design).

```

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

```

```
648 }
```

Now the version when the command is used in the key `caption`. The main difficulty is that the argument of the command `\caption` is composed several times. In order to know the number of commands `\tabularnote` in the caption, we will consider that there should not be the same tabular note twice in the caption (in the main tabular, it's possible). Once we have found a tabular note which has yet been encountered, we consider that you are in a new composition of the argument of `\caption`.

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

Now, we try to detect duplicate notes in the caption. Be careful! We must put `\tl_if_in:NnF` and not `\tl_if_in:NnT`!

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

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

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

Now, we know that are in the second composition of the caption since we are reading a tabular note which has yet been read. Now, the value of `\g_@@_notes_caption_int` won't change anymore: it's the number of uses *without optional argument* of the command `\tabularnote` in the caption.

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

Now, we will compose the label of the footnote (in the caption). Even if we are not in the first composition, we have to compose that label!

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

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

6 Command for creation of rectangle nodes

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

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

```

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

```

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

```

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

```

7 The options

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

```

730 \tl_new:N \l_@@_caption_tl

```

```

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

```

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

```

733 \bool_new:N \l_@@_caption_above_bool

```

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

```

734 \bool_new:N \l_@@_standard_cline_bool

```

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

```

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

```

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

```

737 \bool_new:N \l_@@_xdots_h_labels_bool

```

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

```

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

```

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

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

```

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

```

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

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

```

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

```

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

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

```

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

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Sometimes, we want to have several arrays vertically juxtaposed in order to have an alignment of the columns of these arrays. To achieve this goal, one may wish to use the same width for all the columns (for example with the option `columns-width` or the option `auto-columns-width` of the environment `{NiceMatrixBlock}`). However, even if we use the same type of delimiters, the width of the delimiters may be different from an array to another because the width of the delimiter is fonction of its size. That’s why we create an option called `delimiters/max-width` which will give to the delimiters the width of a delimiter (of the same type) of big size. The following boolean corresponds to this option.

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

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

```

803     {
804         \bool_lazy_or:nnTF
805         { \cs_if_exist_p:N \tikzpicture }
806         { \str_if_eq_p:nn { #1 } { standard } }
807         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
808         { \@@_error:n { bad-option-for-line-style } }
809     } ,
810     line-style .value_required:n = true ,
811     color .tl_set:N = \l_@@_xdots_color_tl ,
812     color .value_required:n = true ,
813     radius .code:n =
814         \hook_gput_code:nnn { begindocument } { . }
815         { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
816     radius .value_required:n = true ,
817     inter .code:n =
818         \hook_gput_code:nnn { begindocument } { . }
819         { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
820     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}`.

```

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

```

824     draw-first .code:n = \prg_do_nothing: ,
825     unknown .code:n = \@@_error:n { Unknown~key~for~xdots }
826 }

```

```

827 \keys_define:nn { nicematrix / rules }
828 {
829     color .tl_set:N = \l_@@_rules_color_tl ,
830     color .value_required:n = true ,
831     width .dim_set:N = \arrayrulewidth ,
832     width .value_required:n = true ,
833     unknown .code:n = \@@_error:n { Unknown~key~for~rules }
834 }
835 \cs_new_protected:Npn \@@_err_key_color_inside:
836 {
837     \@@_warning:n { key~color~inside }
838     \cs_gset:Npn \@@_err_key_color_inside: { }
839 }

```

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

```

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

```

```

853 rounded-corners .default:n = 4 pt ,
854 custom-line .code:n = \@@_custom_line:n { #1 } ,
855 rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
856 rules .value_required:n = true ,
857 standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
858 standard-cline .default:n = true ,
859 cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
860 cell-space-top-limit .value_required:n = true ,
861 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
862 cell-space-bottom-limit .value_required:n = true ,
863 cell-space-limits .meta:n =
864 {
865     cell-space-top-limit = #1 ,
866     cell-space-bottom-limit = #1 ,
867 } ,
868 cell-space-limits .value_required:n = true ,
869 xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
870 light-syntax .code:n =
871     \bool_set_true:N \l_@@_light_syntax_bool
872     \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
873 light-syntax .value_forbidden:n = true ,
874 light-syntax-expanded .code:n =
875     \bool_set_true:N \l_@@_light_syntax_bool
876     \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
877 light-syntax-expanded .value_forbidden:n = true ,
878 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
879 end-of-row .value_required:n = true ,
880 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
881 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
882 last-row .int_set:N = \l_@@_last_row_int ,
883 last-row .default:n = -1 ,
884 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
885 code-for-first-col .value_required:n = true ,
886 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
887 code-for-last-col .value_required:n = true ,
888 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
889 code-for-first-row .value_required:n = true ,
890 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
891 code-for-last-row .value_required:n = true ,
892 hlines .clist_set:N = \l_@@_hlines_clist ,
893 vlines .clist_set:N = \l_@@_vlines_clist ,
894 hlines .default:n = all ,
895 vlines .default:n = all ,
896 vlines-in-sub-matrix .code:n =
897 {
898     \tl_if_single_token:nTF { #1 }
899     {
900         \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
901         { \@@_error:nn { Forbidden-letter } { #1 } }

```

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

```

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

```

```

915     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
916     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
917     \bool_set_true:N \l_@@_hvlines_bool
918     \bool_set_true:N \l_@@_except_borders_bool
919   } ,
920   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.

```

921   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
922   renew-dots .value_forbidden:n = true ,
923   nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
924   create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
925   create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
926   create-extra-nodes .meta:n =
927     { create-medium-nodes , create-large-nodes } ,
928   left-margin .dim_set:N = \l_@@_left_margin_dim ,
929   left-margin .default:n = \arraycolsep ,
930   right-margin .dim_set:N = \l_@@_right_margin_dim ,
931   right-margin .default:n = \arraycolsep ,
932   margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
933   margin .default:n = \arraycolsep ,
934   extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
935   extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
936   extra-margin .meta:n =
937     { extra-left-margin = #1 , extra-right-margin = #1 } ,
938   extra-margin .value_required:n = true ,
939   respect-arraystretch .code:n =
940     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
941   respect-arraystretch .value_forbidden:n = true ,
942   pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
943   pgf-node-code .value_required:n = true
944 }

```

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

```

945 \keys_define:nn { nicematrix / environments }
946 {
947   corners .clist_set:N = \l_@@_corners_clist ,
948   corners .default:n = { NW , SW , NE , SE } ,
949   code-before .code:n =
950     {
951       \tl_if_empty:nF { #1 }
952       {
953         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
954         \bool_set_true:N \l_@@_code_before_bool
955       }
956     } ,
957   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}`.

```

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

```

964   \str_if_eq:eeTF { #1 } { auto }

```

```

965     { \bool_set_true:N \l_@@_auto_columns_width_bool }
966     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
967     columns-width .value_required:n = true ,
968     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.

```

969     \legacy_if:nF { measuring@ }
970     {
971         \str_set:Ne \l_@@_name_str { #1 }
972         \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
973         { \@@_err_duplicate_names:n { #1 } }
974         { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
975     } ,
976     name .value_required:n = true ,
977     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
978     code-after .value_required:n = true ,
979 }

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

982 \keys_define:nn { nicematrix / notes }
983 {
984     para .bool_set:N = \l_@@_notes_para_bool ,
985     para .default:n = true ,
986     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
987     code-before .value_required:n = true ,
988     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
989     code-after .value_required:n = true ,
990     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
991     bottomrule .default:n = true ,
992     style .cs_set:Np = \@@_notes_style:n #1 ,
993     style .value_required:n = true ,
994     label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
995     label-in-tabular .value_required:n = true ,
996     label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
997     label-in-list .value_required:n = true ,
998     enumitem-keys .code:n =
999     {
1000         \hook_gput_code:nnn { begindocument } { . }
1001         {
1002             \IfPackageLoadedT { enumitem }
1003             { \setlist* [ tabularnotes ] { #1 } }
1004         }
1005     } ,
1006     enumitem-keys .value_required:n = true ,
1007     enumitem-keys-para .code:n =
1008     {
1009         \hook_gput_code:nnn { begindocument } { . }
1010         {
1011             \IfPackageLoadedT { enumitem }
1012             { \setlist* [ tabularnotes* ] { #1 } }
1013         }
1014     } ,
1015     enumitem-keys-para .value_required:n = true ,
1016     detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1017     detect-duplicates .default:n = true ,
1018     unknown .code:n = \@@_error:n { Unknown~key~for~notes }
1019 }

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

```



```

1025     color .value_required:n = true ,
1026 }

```

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

```

1027 \keys_define:nn { nicematrix }
1028 {
1029     NiceMatrixOptions .inherit:n =
1030     { nicematrix / Global } ,
1031     NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1032     NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1033     NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1034     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1035     SubMatrix / rules .inherit:n = nicematrix / rules ,
1036     CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1037     CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1038     CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1039     NiceMatrix .inherit:n =
1040     {
1041         nicematrix / Global ,
1042         nicematrix / environments ,
1043     } ,
1044     NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1045     NiceMatrix / rules .inherit:n = nicematrix / rules ,
1046     NiceTabular .inherit:n =
1047     {
1048         nicematrix / Global ,
1049         nicematrix / environments
1050     } ,
1051     NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1052     NiceTabular / rules .inherit:n = nicematrix / rules ,
1053     NiceTabular / notes .inherit:n = nicematrix / notes ,
1054     NiceArray .inherit:n =
1055     {
1056         nicematrix / Global ,
1057         nicematrix / environments ,
1058     } ,
1059     NiceArray / xdots .inherit:n = nicematrix / xdots ,
1060     NiceArray / rules .inherit:n = nicematrix / rules ,
1061     pNiceArray .inherit:n =
1062     {
1063         nicematrix / Global ,
1064         nicematrix / environments ,
1065     } ,
1066     pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1067     pNiceArray / rules .inherit:n = nicematrix / rules ,
1068 }

```

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

```

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

```

```

1081      { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1082      \int_zero:N \l_@@_last_col_int ,
1083      small .bool_set:N = \l_@@_small_bool ,
1084      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.

```

1085      renew-matrix .code:n = \@@_renew_matrix: ,
1086      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}`.

```

1087      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.

```

1088      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)`.

```

1089      \str_if_eq:eeTF { #1 } { auto }
1090      { \@@_error:n { Option-auto-for-columns-width } }
1091      { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1092      allow-duplicate-names .code:n =
1093      \cs_set:Nn \@@_err_duplicate_names:n { } ,
1094      allow-duplicate-names .value_forbidden:n = true ,
1095      notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1096      notes .value_required:n = true ,
1097      sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1098      sub-matrix .value_required:n = true ,
1099      matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1100      matrix / columns-type .value_required:n = true ,
1101      caption-above .bool_set:N = \l_@@_caption_above_bool ,
1102      caption-above .default:n = true ,
1103      unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrixOptions }
1104  }

```

`\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.

```

1105 \NewDocumentCommand \NiceMatrixOptions { m }
1106 { \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.

```

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

```

```

1119   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1120   delimiters / color .value_required:n = true ,
1121   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1122   delimiters / max-width .default:n = true ,
1123   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1124   delimiters .value_required:n = true ,
1125   small .bool_set:N = \l_@@_small_bool ,
1126   small .value_forbidden:n = true ,
1127   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1128 }

```

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

```

1129 \keys_define:nn { nicematrix / NiceArray }
1130 {

```

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.

```

1131   small .bool_set:N = \l_@@_small_bool ,
1132   small .value_forbidden:n = true ,
1133   last-col .code:n = \tl_if_empty:nF { #1 }
1134                     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1135                     \int_zero:N \l_@@_last_col_int ,
1136   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1137   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1138   unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1139 }

1140 \keys_define:nn { nicematrix / pNiceArray }
1141 {
1142   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1143   last-col .code:n = \tl_if_empty:nF { #1 }
1144                     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1145                     \int_zero:N \l_@@_last_col_int ,
1146   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1147   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1148   delimiters / color .value_required:n = true ,
1149   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1150   delimiters / max-width .default:n = true ,
1151   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1152   delimiters .value_required:n = true ,
1153   small .bool_set:N = \l_@@_small_bool ,
1154   small .value_forbidden:n = true ,
1155   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1156   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1157   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1158 }

```

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

```

1159 \keys_define:nn { nicematrix / NiceTabular }
1160 {

```

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.

```

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

```

```

1168   caption .value_required:n = true ,
1169   short-caption .tl_set:N = \l_@@_short_caption_tl ,
1170   short-caption .value_required:n = true ,
1171   label .tl_set:N = \l_@@_label_tl ,
1172   label .value_required:n = true ,
1173   last-col .code:n = \tl_if_empty:nF { #1 }
1174     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1175     \int_zero:N \l_@@_last_col_int ,
1176   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1177   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1178   unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
1179 }

```

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
```

```

1180 \keys_define:nn { nicematrix / CodeAfter }
1181 {
1182   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1183   delimiters / color .value_required:n = true ,
1184   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1185   rules .value_required:n = true ,
1186   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1187   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1188   sub-matrix .value_required:n = true ,
1189   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1190 }

```

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}`).

```

1191 \cs_new_protected:Npn \@@_cell_begin:
1192 {

```

`\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.

```
1193   \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).

```
1194   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1195   \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.

```

1196   \int_compare:nNnT { \c@jCol } = { \c_one_int }
1197   {
1198     \int_compare:nNnT { \l_@@_first_col_int } = { \c_one_int }
1199     { \@@_begin_of_row: }
1200   }

```

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:`.

```
1201 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1202 \@@_tuning_not_tabular_begin:
1203 \@@_tuning_first_row:
1204 \@@_tuning_last_row:
1205 \g_@@_row_style_tl
1206 }
```

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.

```
1207 \cs_new_protected:Npn \@@_tuning_first_row:
1208 {
1209   \if_int_compare:w \c@iRow = \c_zero_int
1210     \if_int_compare:w \c@jCol > \c_zero_int
1211       \l_@@_code_for_first_row_tl
1212       \xglobal \colorlet { nicematrix-first-row } { . }
1213     \fi:
1214   \fi:
1215 }
```

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

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

We will use a version a little more efficient.

```
1216 \cs_new_protected:Npn \@@_tuning_last_row:
1217 {
1218   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1219     \l_@@_code_for_last_row_tl
1220     \xglobal \colorlet { nicematrix-last-row } { . }
1221   \fi:
1222 }
```

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

```
1223 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```

1224 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1225 {
1226   \m@th
1227   \c_math_toggle_token

```

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

```

1228   \@@_tuning_key_small:
1229 }
1230 \cs_set_eq:NN \@@_tuning_not_tabular_end: \c_math_toggle_token

```

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

```

1231 \cs_new_protected:Npn \@@_begin_of_row:
1232 {
1233   \int_gincr:N \c@iRow
1234   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1235   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1236   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1237   \pgfpicture
1238   \pgfrememberpicturepositiononpagetrue
1239   \pgfcoordinate
1240   { \@@_env: - row - \int_use:N \c@iRow - base }
1241   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1242   \str_if_empty:NF \l_@@_name_str
1243   {
1244     \pgfnodealias
1245     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1246     { \@@_env: - row - \int_use:N \c@iRow - base }
1247   }
1248   \endpgfpicture
1249 }

```

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.

```

1250 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1251 {
1252   \int_if_zero:nTF { \c@iRow }
1253   {
1254     \dim_compare:nNnT
1255     { \box_dp:N \l_@@_cell_box } > { \g_@@_dp_row_zero_dim }
1256     { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1257     \dim_compare:nNnT
1258     { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_zero_dim }
1259     { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1260   }
1261   {
1262     \int_compare:nNnT { \c@iRow } = { \c_one_int }
1263     {
1264       \dim_compare:nNnT
1265       { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_one_dim }
1266       { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1267     }
1268   }
1269 }

```

```

1270 \cs_new_protected:Npn \@@_rotate_cell_box:
1271 {
1272   \box_rotate:Nn \l_@@_cell_box { 90 }
1273   \bool_if:NTF \g_@@_rotate_c_bool
1274   {
1275     \hbox_set:Nn \l_@@_cell_box
1276     {
1277       \m@th
1278       \c_math_toggle_token
1279       \vcenter { \box_use:N \l_@@_cell_box }
1280       \c_math_toggle_token
1281     }
1282   }
1283   {
1284     \int_compare:nNtT { \c@iRow } = { \l_@@_last_row_int }
1285     {
1286       \vbox_set_top:Nn \l_@@_cell_box
1287       {
1288         \vbox_to_zero:n { }
1289         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1290         \box_use:N \l_@@_cell_box
1291       }
1292     }
1293   }
1294   \bool_gset_false:N \g_@@_rotate_bool
1295   \bool_gset_false:N \g_@@_rotate_c_bool
1296 }
1297 \cs_new_protected:Npn \@@_adjust_size_box:
1298 {
1299   \dim_compare:nNtT { \g_@@_blocks_wd_dim } > { \c_zero_dim }
1300   {
1301     \box_set_wd:Nn \l_@@_cell_box
1302     { \dim_max:nn { \box_wd:N \l_@@_cell_box } { \g_@@_blocks_wd_dim } }
1303     \dim_gzero:N \g_@@_blocks_wd_dim
1304   }
1305   \dim_compare:nNtT { \g_@@_blocks_dp_dim } > { \c_zero_dim }
1306   {
1307     \box_set_dp:Nn \l_@@_cell_box
1308     { \dim_max:nn { \box_dp:N \l_@@_cell_box } { \g_@@_blocks_dp_dim } }
1309     \dim_gzero:N \g_@@_blocks_dp_dim
1310   }
1311   \dim_compare:nNtT { \g_@@_blocks_ht_dim } > { \c_zero_dim }
1312   {
1313     \box_set_ht:Nn \l_@@_cell_box
1314     { \dim_max:nn { \box_ht:N \l_@@_cell_box } { \g_@@_blocks_ht_dim } }
1315     \dim_gzero:N \g_@@_blocks_ht_dim
1316   }
1317 }
1318 \cs_new_protected:Npn \@@_cell_end:
1319 {

```

The following command is nullified in the tabulars.

```

1320   \@@_tuning_not_tabular_end:
1321   \hbox_set_end:
1322   \@@_cell_end_i:
1323 }
1324 \cs_new_protected:Npn \@@_cell_end_i:
1325 {

```

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.

```

1326   \g_@@_cell_after_hook_tl
1327   \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }

```

```

1328 \@@_adjust_size_box:
1329 \box_set_ht:Nn \l_@@_cell_box
1330 { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1331 \box_set_dp:Nn \l_@@_cell_box
1332 { \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”).

```

1333 \@@_update_max_cell_width:

```

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

```

1334 \@@_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.

```

1335 \bool_if:NTF \g_@@_empty_cell_bool
1336 { \box_use_drop:N \l_@@_cell_box }
1337 {
1338   \bool_if:NTF \g_@@_not_empty_cell_bool
1339   { \@@_print_node_cell: }
1340   {
1341     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
1342     { \@@_print_node_cell: }
1343     { \box_use_drop:N \l_@@_cell_box }
1344   }
1345 }
1346 \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
1347 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1348 \bool_gset_false:N \g_@@_empty_cell_bool
1349 \bool_gset_false:N \g_@@_not_empty_cell_bool
1350 }

```

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

```

1351 \cs_new_protected:Npn \@@_update_max_cell_width:
1352 {
1353   \dim_gset:Nn \g_@@_max_cell_width_dim
1354   { \dim_max:nn { \g_@@_max_cell_width_dim } { \box_wd:N \l_@@_cell_box } }
1355 }

```

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`).

```

1356 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1357 {

```



```

1358 \@@_math_toggle:
1359 \hbox_set_end:
1360 \bool_if:NF \g_@@_rotate_bool
1361 {
1362   \hbox_set:Nn \l_@@_cell_box
1363   {
1364     \makebox [ \l_@@_col_width_dim ] [ s ]
1365     { \hbox_unpack_drop:N \l_@@_cell_box }
1366   }
1367 }
1368 \@@_cell_end_i:
1369 }

1370 \pgfset
1371 {
1372   nicematrix / cell-node /.style =
1373   {
1374     inner-sep = \c_zero_dim ,
1375     minimum-width = \c_zero_dim
1376   }
1377 }

```

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.

```

1378 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1379 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1380 {
1381   \use:c
1382   {
1383     __siunitx_table_align_
1384     \bool_if:NTF \l__siunitx_table_text_bool
1385     { \l__siunitx_table_align_text_tl }
1386     { \l__siunitx_table_align_number_tl }
1387   }
1388   :n
1389   { #1 }
1390 }

```

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!

```

1391 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1392 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1393 {
1394   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1395   \hbox:n
1396   {
1397     \pgfsys@markposition
1398     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1399   }
1400   #1
1401   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1402   \hbox:n
1403   {
1404     \pgfsys@markposition
1405     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1406   }
1407 }

```

```

1408 \cs_new_protected:Npn \@@_print_node_cell:
1409 {
1410   \socket_use:nn { nicematrix / siunitx-wrap }
1411   { \socket_use:nn { nicematrix / create-cell-nodes } \@@_node_cell: }
1412 }

```

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

```

1413 \cs_new_protected:Npn \@@_node_cell:
1414 {
1415   \pgfpicture
1416   \pgfsetbaseline \c_zero_dim
1417   \pgfrememberpicturepositiononpagetrue
1418   \pgfset { nicematrix / cell-node }
1419   \pgfnode
1420   { rectangle }
1421   { base }
1422   {

```

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).

```

1423     \set@color
1424     \box_use:N \l_@@_cell_box
1425   }
1426   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1427   { \l_@@_pgf_node_code_tl }
1428   \str_if_empty:NF \l_@@_name_str
1429   {
1430     \pgfnodealias
1431     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1432     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1433   }
1434   \endpgfpicture
1435 }

```

The second argument of the following command `\@@_instruction_of_type:nnn` defined below is the type of the instruction (`Cdots`, `Vdots`, `Ddots`, etc.). The third argument is the list of options. This command writes in the corresponding `\g_@@_type_lines_tl` the instruction which will actually draw the line after the construction of the matrix.

For example, for the following matrix,

```

\begin{pNiceMatrix}
1 & 2 & 3 & 4 \\
5 & \Cdots & & 6 \\
7 & \Cdots & [color=red] & 
\end{pNiceMatrix}

```

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

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

```

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

```

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

```

1436 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1437 {
1438   \bool_if:nTF { #1 } { \tl_gput_left:ce } { \tl_gput_right:ce }
1439   { g_@@_ #2 _ lines _ tl }
1440   {
1441     \use:c { @@ _ draw _ #2 : nnn }
1442     { \int_use:N \c@iRow }

```

```

1443         { \int_use:N \c@jCol }
1444         { \exp_not:n { #3 } }
1445     }
1446 }

1447 \cs_new_protected:Npn \@@_array:n
1448 {
1449 %     \begin{macrocode}
1450     \dim_set:Nn \col@sep
1451     { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1452     \dim_compare:nNnTF { \l_@@_tabular_width_dim } = { \c_zero_dim }
1453     { \def \@halignto { } }
1454     { \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`.

```

1455     \@tabarray
\l_@@_baseline_tl may have the value t, c or b. However, if the value is b, we compose
the \array (of array) with the option t and the right translation will be done further. Re-
mark that \str_if_eq:eeTF is fully expandable and we need something fully expandable here.
\str_if_eq:ee(TF) is faster than \str_if_eq:nn(TF).

1456     [ \str_if_eq:eeTF \l_@@_baseline_tl { c } { c } { t } ]
1457 }
1458 \cs_generate_variant:Nn \@@_array:n { o }

```

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

```

1459 \bool_if:nTF
1460 { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }

```

We use here a `\cs_set_eq:cN` instead of a `\cs_set_eq:NN` in order to avoid a message when `explcheck` is used on `nicematrix.sty`.

```

1461 { \cs_set_eq:cN { @@_old_ar@ialign: } \ar@ialign }
1462 { \cs_set_eq:NN \@@_old_ialign: \ialign }

```

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

```

1463 \cs_new_protected:Npn \@@_create_row_node:
1464 {
1465     \int_compare:nNnT { \c@iRow } > { \g_@@_last_row_node_int }
1466     {
1467         \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1468         \@@_create_row_node_i:
1469     }
1470 }

1471 \cs_new_protected:Npn \@@_create_row_node_i:
1472 {

```

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

```

1473     \hbox
1474     {
1475         \bool_if:NT \l_@@_code_before_bool
1476         {
1477             \vtop
1478             {
1479                 \skip_vertical:N 0.5\arrayrulewidth
1480                 \pgfsys@markposition
1481                 { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1482                 \skip_vertical:N -0.5\arrayrulewidth
1483             }
1484         }
1485     \pgfpicture

```

```

1486 \pgfrememberpicturepositiononpagetrue
1487 \pgfcoordinate { \l_@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1488 { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1489 \str_if_empty:NF \l_@@_name_str
1490 {
1491   \pgfnodealias
1492   { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1493   { \l_@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1494 }
1495 \endpgfpicture
1496 }
1497 }

1498 \cs_new_protected:Npn \l_@@_in_everycr:
1499 {
1500   \bool_if:NT \c_@@_recent_array_bool
1501   {
1502     \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1503     \tbl_update_cell_data_for_next_row:
1504   }
1505   \int_gzero:N \c@jCol
1506   \bool_gset_false:N \g_@@_after_col_zero_bool
1507   \bool_if:NF \g_@@_row_of_col_done_bool
1508   {
1509     \l_@@_create_row_node:

```

We don't draw now the rules of the key `hlines` (or `hvlines`) but we reserve the vertical space for these rules (the rules will be drawn by PGF).

```

1510 \clist_if_empty:NF \l_@@_hlines_clist
1511 {
1512   \str_if_eq:eeF \l_@@_hlines_clist { all }
1513   {
1514     \clist_if_in:NeT
1515     \l_@@_hlines_clist
1516     { \int_eval:n { \c@iRow + 1 } }
1517   }
1518   {

```

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.

```

1519 \int_compare:nNnT { \c@iRow } > { -1 }
1520 {
1521   \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
1522   { \hrule height \arrayrulewidth width \c_zero_dim }
1523 }
1524 }
1525 }
1526 }
1527 }

```

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

```

1528 \cs_set_protected:Npn \l_@@_renew_dots:
1529 {
1530   \cs_set_eq:NN \ldots \l_@@_Ldots:
1531   \cs_set_eq:NN \cdots \l_@@_Cdots:
1532   \cs_set_eq:NN \vdots \l_@@_Vdots:
1533   \cs_set_eq:NN \ddots \l_@@_Ddots:
1534   \cs_set_eq:NN \iddots \l_@@_Iddots:
1535   \cs_set_eq:NN \dots \l_@@_Ldots:
1536   \cs_set_eq:NN \hdotsfor \l_@@_Hdotsfor:
1537 }

```

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 redefinition⁵.

```

1538 \hook_gput_code:nnn { begindocument } { . }
1539 {
1540   \IfPackageLoadedTF { booktabs }
1541   {
1542     \cs_new_protected:Npn \@_patch_booktabs:
1543     { \tl_put_left:Nn \@BTnormal \@_create_row_node_i: }
1544   }
1545   { \cs_new_protected:Npn \@_patch_booktabs: { } }
1546 }

```

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`⁶ 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`.

```

1547 \cs_new_protected:Npn \@_some_initialization:
1548 {
1549   \@_everycr:
1550   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1551   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1552   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1553   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1554   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1555   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1556 }

```

```

1557 \cs_new_protected:Npn \@_pre_array_ii:
1558 {

```

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

```

1559   \fp_gzero:N \g_@@_total_X_weight_fp
1560   \@_expand_clist:N \l_@@_hlines_clist
1561   \@_expand_clist:N \l_@@_vlines_clist
1562   \@_patch_booktabs:
1563   \box_clear_new:N \l_@@_cell_box
1564   \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}`).

```

1565   \bool_if:NT \l_@@_small_bool
1566   {
1567     \def \arraystretch { 0.47 }
1568     \dim_set:Nn \arraycolsep { 1.45 pt }

```

⁵cf. `\nicematrix@redefine@check@rerun`

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

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

```
1569     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1570 }
```

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!

```
1571 \bool_if:NT \g_@@_create_cell_nodes_bool
1572 {
1573     \tl_put_right:Nn \@@_begin_of_row:
1574     {
1575         \pgfsys@markposition
1576         { \@@_env: - row - \int_use:N \c@iRow - base }
1577     }
1578     \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1579 }
```

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

```
1580 \bool_if:nTF
1581 { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }
1582 {
1583     \def \ar@ialign
1584     {
1585         \bool_if:NT \c_@@_testphase_table_bool
1586         \tbl_init_cell_data_for_table:
1587         \@@_some_initialization:
1588         \dim_zero:N \tabskip
1589     }
1590 }
```

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`.

```
1589     \cs_set_eq:Nc \ar@ialign { @@_old_ar@ialign: }
1590     \halign
1591 }
1592 }
```

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

```
1593 {
1594     \def \ialign
1595     {
1596         \@@_some_initialization:
1597         \dim_zero:N \tabskip
1598         \cs_set_eq:NN \ialign \@@_old_ialign:
1599         \halign
1600     }
1601 }
```

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).

```
1602 \bool_if:NT \c_@@_revtex_bool
1603 {
1604     \IfPackageLoadedT { colortbl }
1605     { \cs_set_protected:Npn \CT@setup { } }
1606 }
```

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).

```

1607 \cs_set_eq:NN \@@_old_ldots: \ldots
1608 \cs_set_eq:NN \@@_old_cdots: \cdots
1609 \cs_set_eq:NN \@@_old_vdots: \vdots
1610 \cs_set_eq:NN \@@_old_ddots: \ddots
1611 \cs_set_eq:NN \@@_old_iddots: \iddots
1612 \bool_if:NTF \l_@@_standard_cline_bool
1613 { \cs_set_eq:NN \cline \@@_standard_cline: }
1614 { \cs_set_eq:NN \cline \@@_cline: }
1615 \cs_set_eq:NN \Ldots \@@_Ldots:
1616 \cs_set_eq:NN \Cdots \@@_Cdots:
1617 \cs_set_eq:NN \Vdots \@@_Vdots:
1618 \cs_set_eq:NN \Ddots \@@_Ddots:
1619 \cs_set_eq:NN \Iddots \@@_Iddots:
1620 \cs_set_eq:NN \Hline \@@_Hline:
1621 \cs_set_eq:NN \Hspace \@@_Hspace:
1622 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1623 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1624 \cs_set_eq:NN \Block \@@_Block:
1625 \cs_set_eq:NN \rotate \@@_rotate:
1626 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1627 \cs_set_eq:NN \dotfill \@@_dotfill:
1628 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1629 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1630 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1631 \cs_set_eq:NN \TopRule \@@_TopRule
1632 \cs_set_eq:NN \MidRule \@@_MidRule
1633 \cs_set_eq:NN \BottomRule \@@_BottomRule
1634 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1635 \cs_set_eq:NN \Hbrace \@@_Hbrace
1636 \cs_set_eq:NN \Vbrace \@@_Vbrace
1637 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1638 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1639 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1640 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1641 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1642 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1643 \int_compare:nNt { \l_@@_first_row_int } > { \c_zero_int }
1644 { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1645 \int_compare:nNt { \l_@@_last_row_int } < { \c_zero_int }
1646 { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1647 \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:`.

```

1648 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1649 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1650 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1651 \@@_revert_colortbl:

```

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

```

1652 \tl_if_exist:NT \l_@@_note_in_caption_tl
1653 {
1654   \tl_if_empty:NF \l_@@_note_in_caption_tl
1655   {
1656     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl

```

```

1657         \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1658     }
1659 }

```

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).

```

1660     \seq_gclear:N \g_@@_multicolumn_cells_seq
1661     \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).

```

1662     \int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }

```

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

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

```

1663     \int_gzero_new:N \g_@@_row_total_int

```

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

```

1664     \int_gzero_new:N \g_@@_col_total_int
1665     \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1666     \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.

```

1667     \tl_gclear_new:N \g_@@_Cdots_lines_tl
1668     \tl_gclear_new:N \g_@@_Ldots_lines_tl
1669     \tl_gclear_new:N \g_@@_Vdots_lines_tl
1670     \tl_gclear_new:N \g_@@_Ddots_lines_tl
1671     \tl_gclear_new:N \g_@@_Iddots_lines_tl
1672     \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1673     \tl_gclear:N \g_nicematrix_code_before_tl
1674     \tl_gclear:N \g_@@_pre_code_before_tl
1675 }

```

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

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

```

1676 \cs_new_protected:Npn \@@_pre_array:
1677 {
1678     \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1679     \int_gzero_new:N \c@iRow
1680     \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1681     \int_gzero_new:N \c@jCol

```

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

```

1682     \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
1683     {
1684         \bool_set_true:N \l_@@_last_row_without_value_bool
1685         \bool_if:NT \g_@@_aux_found_bool
1686         { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1687     }

```



```

1688 \int_compare:nNt { \l_@@_last_col_int } = { -1 }
1689 {
1690     \bool_if:NT \g_@@_aux_found_bool
1691     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1692 }

```

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”.

```

1693 \int_compare:nNt { \l_@@_last_row_int } > { -2 }
1694 {
1695     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1696     {
1697         \dim_compare:nNt { \g_@@_ht_last_row_dim } < { \box_ht:N \l_@@_cell_box }
1698         { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1699         \dim_compare:nNt { \g_@@_dp_last_row_dim } < { \box_dp:N \l_@@_cell_box }
1700         { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1701     }
1702 }

1703 \seq_gclear:N \g_@@_cols_vlism_seq
1704 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1705 \bool_if:NT \l_@@_code_before_bool { \@@_exec_code_before: }

```

The value of `\g_@@_pos_of_blocks_seq` has been written on the `aux` file and loaded before the (potential) execution of the `\CodeBefore`. Now, we clear that variable because it will be reconstructed during the creation of the array.

```

1706 \seq_gset_eq:NN \g_@@_pos_of_blocks_seq \g_@@_future_pos_of_blocks_seq
1707 \seq_gclear:N \g_@@_future_pos_of_blocks_seq

```

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

```

1708 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1709 \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.

```

1710 \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1711 \@@_pre_array_ii:

```

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

```

1712 \box_clear_new:N \l_@@_the_array_box

```

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

```

1713 \dim_zero_new:N \l_@@_left_delim_dim
1714 \dim_zero_new:N \l_@@_right_delim_dim
1715 \bool_if:NTF \g_@@_delims_bool
1716 {

```

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

```

1717     \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1718     \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1719     \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1720     \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1721   }
1722   {
1723     \dim_gset:Nn \l_@@_left_delim_dim
1724       { 2 \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1725     \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1726   }

```

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

```

1727   \hbox_set:Nw \l_@@_the_array_box
1728   \skip_horizontal:N \l_@@_left_margin_dim
1729   \skip_horizontal:N \l_@@_extra_left_margin_dim
1730   \bool_if:NT \c_@@_recent_array_bool
1731     { \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).

```

1732   \m@th
1733   \c_math_toggle_token
1734   \bool_if:NTF \l_@@_light_syntax_bool
1735     { \use:c { @@-light-syntax } }
1736     { \use:c { @@-normal-syntax } }
1737 }

```

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

```

1738 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1739 {
1740   \tl_set:Nn \l_tmpa_tl { #1 }
1741   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1742     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1743   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1744   \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...

```

1745   \@@_pre_array:
1746 }

```

9 The `\CodeBefore`

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

```

1747 \cs_new_protected:Npn \@@_pre_code_before:
1748 {

```

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

```

1749 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1750 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1751 \int_set:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1752 \int_set:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }

```

Now, we will create all the `col` nodes and `row` nodes with the information written in the `aux` file. You use the technique described in the page 1247 of `pgfmanual.pdf`, version 3.1.10.

```

1753 \pgfsys@markposition { \@@_env: - position }
1754 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1755 \pgfpicture
1756 \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1757 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int + 1 }
1758 {
1759   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1760   \pgfcoordinate { \@@_env: - row - ##1 }
1761   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1762 }

```

Now, the recreation of the `col` nodes.

```

1763 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int + 1 }
1764 {
1765   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1766   \pgfcoordinate { \@@_env: - col - ##1 }
1767   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1768 }

```

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

```

1769 \@@_create_diag_nodes:

```

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

```

1770 \bool_if:NT \g_@@_create_cell_nodes_bool { \@@_recreate_cell_nodes: }
1771 \endpgfpicture

```

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

```

1772 \@@_create_blocks_nodes:
1773 \IfPackageLoadedT { tikz }
1774 {
1775   \tikzset
1776   {
1777     every-picture / .style =
1778     { overlay , name~prefix = \@@_env: - }
1779   }
1780 }
1781 \cs_set_eq:NN \cellcolor \@@_cellcolor
1782 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1783 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1784 \cs_set_eq:NN \rowcolor \@@_rowcolor
1785 \cs_set_eq:NN \rowcolors \@@_rowcolors
1786 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1787 \cs_set_eq:NN \arraycolor \@@_arraycolor
1788 \cs_set_eq:NN \columncolor \@@_columncolor
1789 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1790 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1791 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1792 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1793 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n

```

```

1794 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1795 }

```

```

1796 \cs_new_protected:Npn \@@_exec_code_before:
1797 {

```

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...

```

1798 \clist_map_inline:Nn \l_@@_corners_cells_clist
1799 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1800 \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`.

```

1801 \@@_add_to_colors_seq:nn { { nocolor } } { }
1802 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1803 \group_begin:

```

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

```

1804 \if_mode_math:
1805 \@@_exec_code_before_i:
1806 \else:
1807 \c_math_toggle_token
1808 \@@_exec_code_before_i:
1809 \c_math_toggle_token
1810 \fi:
1811 \group_end:
1812 }

```

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`).

```

1813 \cs_new_protected:Npn \@@_exec_code_before_i:
1814 {
1815 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1816 { \@@_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`.

```

1817 \exp_last_unbraced:No \@@_CodeBefore_keys:
1818 \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.

```

1819 \@@_actually_color:
1820 \l_@@_code_before_tl
1821 \q_stop
1822 }

```

```

1823 \keys_define:nn { nicematrix / CodeBefore }
1824 {
1825 create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1826 create-cell-nodes .default:n = true ,
1827 sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1828 sub-matrix .value_required:n = true ,

```

```

1829     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1830     delimiters / color .value_required:n = true ,
1831     unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1832 }
1833 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1834 {
1835     \keys_set:nn { nicematrix / CodeBefore } { #1 }
1836     \@@_CodeBefore:w
1837 }

```

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.

```

1838 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1839 {
1840     \bool_if:NT \g_@@_aux_found_bool
1841     {
1842         \@@_pre_code_before:
1843         \legacy_if:nF { measuring@ } { #1 }
1844     }
1845 }

```

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.

```

1846 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1847 {
1848     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
1849     {
1850         \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1851         \pgfcoordinate { \@@_env: - row - ##1 - base }
1852         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1853         \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
1854         {
1855             \cs_if_exist:cT
1856             { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1857             {
1858                 \pgfsys@getposition
1859                 { \@@_env: - ##1 - #####1 - NW }
1860                 \@@_node_position:
1861                 \pgfsys@getposition
1862                 { \@@_env: - ##1 - #####1 - SE }
1863                 \@@_node_position_i:
1864                 \@@_pgf_rect_node:nnn
1865                 { \@@_env: - ##1 - #####1 }
1866                 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1867                 { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1868             }
1869         }
1870     }
1871     \@@_create_extra_nodes:
1872     \@@_create_aliases_last:
1873 }
1874 \cs_new_protected:Npn \@@_create_aliases_last:
1875 {
1876     \int_step_inline:nn { \c@iRow }
1877     {
1878         \pgfnodealias
1879         { \@@_env: - ##1 - last }
1880         { \@@_env: - ##1 - \int_use:N \c@jCol }

```

```

1881     }
1882     \int_step_inline:nn { \c@jCol }
1883     {
1884         \pgfnodealias
1885         { \@@_env: - last - ##1 }
1886         { \@@_env: - \int_use:N \c@iRow - ##1 }
1887     }
1888     \pgfnodealias % added 2025-04-05
1889     { \@@_env: - last - last }
1890     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1891 }

```

```

1892 \cs_new_protected:Npn \@@_create_blocks_nodes:
1893 {
1894     \pgfpicture
1895     \pgf@relevantforpicturesizefalse
1896     \pgfrememberpicturepositiononpagetrue
1897     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1898     { \@@_create_one_block_node:nnnnn ##1 }
1899     \endpgfpicture
1900 }

```

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.⁷

```

1901 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1902 {
1903     \tl_if_empty:nF { #5 }
1904     {
1905         \@@_qpoint:n { col - #2 }
1906         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1907         \@@_qpoint:n { #1 }
1908         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1909         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1910         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1911         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1912         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1913         \@@_pgf_rect_node:nnnnn
1914         { \@@_env: - #5 }
1915         { \dim_use:N \l_tmpa_dim }
1916         { \dim_use:N \l_tmpb_dim }
1917         { \dim_use:N \l_@@_tmpc_dim }
1918         { \dim_use:N \l_@@_tmpd_dim }
1919     }
1920 }

```

```

1921 \cs_new_protected:Npn \@@_patch_for_revtex:
1922 {
1923     \cs_set_eq:NN \@addamp \@addamp@LaTeX
1924     \cs_set_eq:NN \@array \@array@array
1925     \cs_set_eq:NN \@tabular \@tabular@array
1926     \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1927     \cs_set_eq:NN \array \array@array
1928     \cs_set_eq:NN \endarray \endarray@array
1929     \cs_set:Npn \endtabular { \endarray $\egroup } % $
1930     \cs_set_eq:NN \@mkpream \@mkpream@array
1931     \cs_set_eq:NN \@classx \@classx@array
1932     \cs_set_eq:NN \insert@column \insert@column@array
1933     \cs_set_eq:NN \@arraycr \@arraycr@array

```

⁷Moreover, 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).

```

1934 \cs_set_eq:NN \@xarraycr \@xarraycr@array
1935 \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1936 }

```

10 The environment `{NiceArrayWithDelims}`

```

1937 \NewDocumentEnvironment { NiceArrayWithDelims }
1938 { m m O { } m ! O { } t \CodeBefore }
1939 {
1940 \bool_if:NT \c_@@_revtex_bool { \@@_patch_for_revtex: }
1941 \@@_provide_pgfsyspdfmark:
1942 \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.

```

1943 \bgroup

1944 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1945 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1946 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1947 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

1948 \int_gzero:N \g_@@_block_box_int
1949 \dim_gzero:N \g_@@_width_last_col_dim
1950 \dim_gzero:N \g_@@_width_first_col_dim
1951 \bool_gset_false:N \g_@@_row_of_col_done_bool
1952 \str_if_empty:NT \g_@@_name_env_str
1953 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1954 \bool_if:NTF \l_@@_tabular_bool
1955 { \mode_leave_vertical: }
1956 { \@@_test_if_math_mode: }
1957 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1958 \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁸. 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.

```

1959 \cs_gset_eq:cN { @@_old_CT@arc@ } \CT@arc@

```

We deactivate Tikz externalization because we will use PGF pictures with the options `overlay` and `remember picture` (or equivalent forms). We deactivate with `\tikzexternaldisable` and not with `\tikzset{external/export=false}` which is *not* equivalent.

```

1960 \cs_if_exist:NT \tikz@library@external@loaded
1961 {
1962 \tikzexternaldisable
1963 \cs_if_exist:NT \ifstandalone
1964 { \tikzset { external / optimize = false } }
1965 }

```

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

```

1966 \int_gincr:N \g_@@_env_int
1967 \bool_if:NF \l_@@_block_auto_columns_width_bool
1968 { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

⁸e.g. `\color[rgb]{0.5,0.5,0}`

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

```
1969 \seq_gclear:N \g_@@_blocks_seq
1970 \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`.

```
1971 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1972 \seq_gclear:N \g_@@_pos_of_xdots_seq
1973 \tl_gclear_new:N \g_@@_code_before_tl
1974 \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.

```
1975 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1976 {
1977   \bool_gset_true:N \g_@@_aux_found_bool
1978   \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1979 }
1980 { \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.

```
1981 \tl_gclear:N \g_@@_aux_tl
1982 \tl_if_empty:NF \g_@@_code_before_tl
1983 {
1984   \bool_set_true:N \l_@@_code_before_bool
1985   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1986 }
1987 \tl_if_empty:NF \g_@@_pre_code_before_tl
1988 { \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`.

```
1989 \bool_if:NTF \g_@@_delims_bool
1990 { \keys_set:nn { nicematrix / pNiceArray } }
1991 { \keys_set:nn { nicematrix / NiceArray } }
1992 { #3 , #5 }
```

```
1993 \@@_set_CTarc:o \l_@@_rules_color_tl
```

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

```
1994 \bool_if:nTF { #6 } { \@@_CodeBefore_Body:w } { \@@_pre_array: }
1995 }
```

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

```
1996 {
1997   \bool_if:NTF \l_@@_light_syntax_bool
1998   { \use:c { end @@-light-syntax } }
1999   { \use:c { end @@-normal-syntax } }
2000   \c_math_toggle_token
2001   \skip_horizontal:N \l_@@_right_margin_dim
2002   \skip_horizontal:N \l_@@_extra_right_margin_dim
2003   \hbox_set_end:
2004   \bool_if:NT \c_@@_recent_array_bool
2005   { \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.

```

2006   \bool_if:NT \l_@@_width_used_bool
2007   {
2008     \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2009     { \@@_error_or_warning:n { width~without~X~columns } }
2010   }

```

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 .

```

2011   \fp_compare:nNnT { \g_@@_total_X_weight_fp } > { \c_zero_fp }
2012   { \@@_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).

```

2013   \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
2014   {
2015     \bool_if:NF \l_@@_last_row_without_value_bool
2016     {
2017       \int_compare:nNnF { \l_@@_last_row_int } = { \c@iRow }
2018       {
2019         \@@_error:n { Wrong~last~row }
2020         \int_gset_eq:NN \l_@@_last_row_int \c@iRow
2021       }
2022     }
2023   }

```

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

```

2024   \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2025   \bool_if:NTF \g_@@_last_col_found_bool
2026   { \int_gdecr:N \c@jCol }
2027   {
2028     \int_compare:nNnT { \l_@@_last_col_int } > { -1 }
2029     { \@@_error:n { last~col~not~used } }
2030   }

```

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

```

2031   \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2032   \int_compare:nNnT { \l_@@_last_row_int } > { -1 }
2033   { \int_gdecr:N \c@iRow }

```

Now, we begin the real construction in the output flow of TeX. First, we take into account a potential “first column” (we remind that this “first column” has been constructed in an overlapping position and that we have computed its width in `\g_@@_width_first_col_dim`: see p. 91).

```

2034   \int_if_zero:nT { \l_@@_first_col_int }
2035   { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2036   \bool_if:nTF { ! \g_@@_delims_bool }
2037   {
2038     \str_if_eq:eeTF \l_@@_baseline_tl { c }
2039     { \@@_use_arraybox_with_notes_c: }
2040     {
2041       \str_if_eq:eeTF \l_@@_baseline_tl { b }
2042       { \@@_use_arraybox_with_notes_b: }
2043       { \@@_use_arraybox_with_notes: }
2044     }
2045   }

```

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

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).

```

2046 {
2047   \int_if_zero:nTF { \l_@@_first_row_int }
2048   {
2049     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2050     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2051   }
2052   { \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”.¹⁰

```

2053   \int_compare:nNnTF { \l_@@_last_row_int } > { -2 }
2054   {
2055     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2056     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2057   }
2058   { \dim_zero:N \l_tmpb_dim }
2059   \hbox_set:Nn \l_tmpa_box
2060   {
2061     \m@th
2062     \c_math_toggle_token
2063     \@@_color:o \l_@@_delimiters_color_tl
2064     \exp_after:wN \left \g_@@_left_delim_tl
2065     \vcenter
2066     {

```

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.

```

2067       \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2068       \hbox
2069       {
2070         \bool_if:NTF \l_@@_tabular_bool
2071         { \skip_horizontal:n { - \tabcolsep } }
2072         { \skip_horizontal:n { - \arraycolsep } }
2073         \@@_use_arraybox_with_notes_c:
2074         \bool_if:NTF \l_@@_tabular_bool
2075         { \skip_horizontal:n { - \tabcolsep } }
2076         { \skip_horizontal:n { - \arraycolsep } }
2077       }

```

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

```

2078       \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2079     }
2080     \exp_after:wN \right \g_@@_right_delim_tl
2081     \c_math_toggle_token
2082   }

```

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.

```

2083   \bool_if:NTF \l_@@_delimiters_max_width_bool
2084   {
2085     \@@_put_box_in_flow_bis:nn
2086     { \g_@@_left_delim_tl }
2087     { \g_@@_right_delim_tl }
2088   }
2089   \@@_put_box_in_flow:
2090 }

```

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. 91).

¹⁰A 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).

```

2091 \bool_if:NT \g_@@_last_col_found_bool
2092   { \skip_horizontal:N \g_@@_width_last_col_dim }
2093 \bool_if:NT \l_@@_preamble_bool
2094   {
2095     \int_compare:nNt { \c@jCol } < { \g_@@_static_num_of_col_int }
2096     { \@@_err_columns_not_used: }
2097   }
2098 \@@_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.

```

2099 \egroup

```

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

```

2100 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2101 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2102 \iow_now:Ne \@mainaux
2103   {
2104     \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2105     \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2106     { \exp_not:o \g_@@_aux_tl }
2107   }
2108 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2109 \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2110 }

```

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

```

2111 \cs_new_protected:Npn \@@_err_columns_not_used:
2112   {
2113     \@@_warning:n { columns~not~used }
2114     \cs_gset:Npn \@@_err_columns_not_used: { }
2115   }

```

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 .

```

2116 \cs_new_protected:Npn \@@_compute_width_X:
2117   {
2118     \tl_gput_right:Ne \g_@@_aux_tl
2119     {
2120       \bool_set_true:N \l_@@_X_columns_aux_bool
2121       \dim_set:Nn \l_@@_X_columns_dim
2122       {
2123         \dim_compare:nNtF
2124         {
2125           \dim_abs:n
2126           { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2127         }
2128         <
2129         { 0.001 pt }
2130         { \dim_use:N \l_@@_X_columns_dim }
2131         {
2132           \dim_eval:n
2133           {
2134             \fp_to_dim:n
2135             {
2136               (
2137                 \dim_eval:n
2138                 { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2139               )

```

```

2140          / \fp_use:N \g_@@_total_X_weight_fp
2141        }
2142      + \l_@@_X_columns_dim
2143    }
2144  }
2145}
2146}
2147}

```

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`.

```

2148 \cs_new_protected:Npn \@@_transform_preamble:
2149 {
2150   \@@_transform_preamble_i:
2151   \@@_transform_preamble_ii:
2152 }
2153 \cs_new_protected:Npn \@@_transform_preamble_i:
2154 {
2155   \int_gzero:N \c@jCol

```

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

```

2156   \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.

```

2157   \bool_gset_false:N \g_tmpb_bool

```

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

```

2158   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2159   \int_zero:N \l_tmpa_int
2160   \tl_gclear:N \g_@@_array_preamble_tl
2161   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2162   {
2163     \tl_gset:Nn \g_@@_array_preamble_tl
2164     { ! { \skip_horizontal:N \arrayrulewidth } }
2165   }
2166   {
2167     \clist_if_in:NnT \l_@@_vlines_clist 1
2168     {
2169       \tl_gset:Nn \g_@@_array_preamble_tl
2170       { ! { \skip_horizontal:N \arrayrulewidth } }
2171     }
2172   }

```

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

```

2173   \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2174   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
2175   \@@_replace_columncolor:
2176 }

```

```

2177 \cs_new_protected:Npn \@@_transform_preamble_ii:
2178 {

```

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

```

2179 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2180 {
2181 \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2182 { \bool_gset_true:N \g_@@_delims_bool }
2183 }
2184 { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2185 \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).

```

2186 \int_if_zero:nTF { \l_@@_first_col_int }
2187 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2188 {
2189 \bool_if:NF \g_@@_delims_bool
2190 {
2191 \bool_if:NF \l_@@_tabular_bool
2192 {
2193 \clist_if_empty:NT \l_@@_vlines_clist
2194 {
2195 \bool_if:NF \l_@@_exterior_arraycolsep_bool
2196 { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2197 }
2198 }
2199 }
2200 }
2201 \int_compare:nNnTF { \l_@@_last_col_int } > { -1 }
2202 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2203 {
2204 \bool_if:NF \g_@@_delims_bool
2205 {
2206 \bool_if:NF \l_@@_tabular_bool
2207 {
2208 \clist_if_empty:NT \l_@@_vlines_clist
2209 {
2210 \bool_if:NF \l_@@_exterior_arraycolsep_bool
2211 { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2212 }
2213 }
2214 }
2215 }

```

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

```

2216 \dim_compare:nNnT { \l_@@_tabular_width_dim } = { \c_zero_dim }
2217 {

```

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

```

2218 \bool_if:NF \c_@@_testphase_table_bool
2219 {
2220 \tl_gput_right:Nn \g_@@_array_preamble_tl
2221 { > { \@@_error_too_much_cols: } 1 }
2222 }
2223 }
2224 }

```

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.

```
2225 \cs_new_protected:Npn \@@_rec_preamble:n #1
2226 {
```

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.¹¹

```
2227 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2228 { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2229 {
```

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

```
2230 \cs_if_exist:cTF { NC @ find @ #1 }
2231 {
2232 \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2233 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2234 }
2235 {
2236 \str_if_eq:nnTF { #1 } { S }
2237 { \@@_fatal:n { unknown~column~type~S } }
2238 { \@@_fatal:nn { unknown~column~type } { #1 } }
2239 }
2240 }
2241 }
```

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

```
2242 \cs_new_protected:Npn \@@_c: #1
2243 {
2244 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2245 \tl_gclear:N \g_@@_pre_cell_tl
2246 \tl_gput_right:Nn \g_@@_array_preamble_tl
2247 { > \@@_cell_begin: c < \@@_cell_end: }
```

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

```
2248 \int_gincr:N \c@jCol
2249 \@@_rec_preamble_after_col:n
2250 }

2251 \cs_new_protected:Npn \@@_l: #1
2252 {
2253 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2254 \tl_gclear:N \g_@@_pre_cell_tl
2255 \tl_gput_right:Nn \g_@@_array_preamble_tl
2256 {
2257 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2258 l
2259 < \@@_cell_end:
2260 }
2261 \int_gincr:N \c@jCol
2262 \@@_rec_preamble_after_col:n
2263 }

2264 \cs_new_protected:Npn \@@_r: #1
2265 {
2266 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2267 \tl_gclear:N \g_@@_pre_cell_tl
2268 \tl_gput_right:Nn \g_@@_array_preamble_tl
2269 {
2270 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
```

¹¹We 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`.

```

2271         r
2272         < \@@_cell_end:
2273     }
2274     \int_gincr:N \c@jCol
2275     \@@_rec_preamble_after_col:n
2276 }

```

For ! and @

```

2277 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2278 {
2279     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2280     \@@_rec_preamble:n
2281 }
2282 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2283 \cs_new_protected:cpn { @@ _ | : } #1
2284 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2285     \int_incr:N \l_tmpa_int
2286     \@@_make_preamble_i_i:n
2287 }
2288 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2289 {
2290     \str_if_eq:nnTF { #1 } { | }
2291     { \use:c { @@ _ | : } | }
2292     { \@@_make_preamble_i_ii:nn { } #1 }
2293 }
2294 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2295 {
2296     \str_if_eq:nnTF { #2 } { [ ] }
2297     { \@@_make_preamble_i_iii:nw { #1 } [ ] }
2298     { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2299 }
2300 \cs_new_protected:Npn \@@_make_preamble_i_iii:nw #1 [ #2 ]
2301 { \@@_make_preamble_i_iii:nn { #1 , #2 } }
2302 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2303 {
2304     \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2305     \tl_gput_right:Ne \g_@@_array_preamble_tl
2306     {

```

Here, the command \dim_use:N is mandatory.

```

2307         \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2308     }
2309     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2310     {
2311         \@@_vline:n
2312         {
2313             position = \int_eval:n { \c@jCol + 1 } ,
2314             multiplicity = \int_use:N \l_tmpa_int ,
2315             total-width = \dim_use:N \l_@@_rule_width_dim ,
2316             #2
2317         }

```

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.

```

2318     }
2319     \int_zero:N \l_tmpa_int
2320     \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2321     \@@_rec_preamble:n #1
2322 }

```

```

2323 \cs_new_protected:cpn { @@ _ > : } #1 #2
2324 {
2325   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2326   \@@_rec_preamble:n
2327 }
2328 \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.

```

2329 \keys_define:nn { nicematrix / p-column }
2330 {
2331   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2332   r .value_forbidden:n = true ,
2333   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2334   c .value_forbidden:n = true ,
2335   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2336   l .value_forbidden:n = true ,
2337   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2338   S .value_forbidden:n = true ,
2339   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2340   p .value_forbidden:n = true ,
2341   t .meta:n = p ,
2342   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2343   m .value_forbidden:n = true ,
2344   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2345   b .value_forbidden:n = true
2346 }

```

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

```

2347 \cs_new_protected:Npn \@@_p: #1
2348 {
2349   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2350   \@@_make_preamble_ii_i:n
2351 }
2352 \cs_set_eq:NN \@@_b: \@@_p:
2353 \cs_set_eq:NN \@@_m: \@@_p:
2354 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2355 {
2356   \str_if_eq:nnTF { #1 } { [ ]
2357     { \@@_make_preamble_ii_ii:w [ ]
2358       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2359     }
2360   \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2361     { \@@_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.

```

2362 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2363 {

```

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).

```

2364   \str_set:Nn \l_@@_hpos_col_str { j }
2365   \@@_keys_p_column:n { #1 }
2366   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2367 }
2368 \cs_new_protected:Npn \@@_keys_p_column:n #1
2369 { \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.

```

2370 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2371 {
2372   \use:e
2373   {
2374     \@@_make_preamble_ii_v:nnnnnnnn
2375     { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }
2376     { \dim_eval:n { #1 } }
2377     {

```

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.

```

2378       \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2379       { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2380       {

```

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

```

2381         \def \exp_not:N \l_@@_hpos_cell_tl
2382         { \str_lowercase:f { \l_@@_hpos_col_str } }
2383       }
2384     \IfPackageLoadedTF { ragged2e }
2385     {
2386       \str_case:on \l_@@_hpos_col_str
2387       {

```

The following `\exp_not:N` are mandatory.

```

2388         c { \exp_not:N \Centering }
2389         l { \exp_not:N \RaggedRight }
2390         r { \exp_not:N \RaggedLeft }
2391       }
2392     }
2393     {
2394       \str_case:on \l_@@_hpos_col_str
2395       {
2396         c { \exp_not:N \centering }
2397         l { \exp_not:N \raggedright }
2398         r { \exp_not:N \raggedleft }
2399       }
2400     }
2401     #3
2402   }
2403   { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2404   { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2405   { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2406   { #2 }
2407   {
2408     \str_case:onF \l_@@_hpos_col_str
2409     {
2410       { j } { c }
2411       { si } { c }
2412     }

```

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

```

2413       { \str_lowercase:f \l_@@_hpos_col_str }
2414     }
2415   }

```

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

```

2416     \int_gincr:N \c@jCol
2417     \@@_rec_preamble_after_col:n
2418   }

```

#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*.

```

2419 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2420 {
2421   \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2422   {
2423     \tl_gput_right:Nn \g_@@_array_preamble_tl
2424     { > \@@_test_if_empty_for_S: }
2425   }
2426   { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2427   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2428   \tl_gclear:N \g_@@_pre_cell_tl
2429   \tl_gput_right:Nn \g_@@_array_preamble_tl
2430   {
2431     > {

```

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.

```

2432   \dim_set:Nn \l_@@_col_width_dim { #2 }
2433   \bool_if:NT \c_@@_testphase_table_bool
2434   { \tag_struct_begin:n { tag = Div } }
2435   \@@_cell_begin:

```

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

```

2436   \use:c { #7 } [ #1 ] { #2 }

```

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

```

2437   \everypar
2438   {
2439     \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2440     \everypar { }
2441   }
2442   \bool_if:NT \c_@@_testphase_table_bool { \tagpdfparaOn }

```

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

```

2443   #3

```

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

```

2444   \g_@@_row_style_tl
2445   \arraybackslash
2446   #5
2447   }
2448   #8
2449   < {
2450   #6

```

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

```

2451   \@finalstrut \@arstrutbox
2452   \use:c { end #7 }

```

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

```

2453         #4
2454         \@@_cell_end:
2455         \bool_if:NT \c_@@_testphase_table_bool { \tag_struct_end: }
2456     }
2457 }
2458 }

```

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

```

2459 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2460 {

```

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 &).

```

2461     \group_align_safe_begin:
2462     \peek_meaning:NTF &
2463     { \@@_the_cell_is_empty: }
2464     {
2465         \peek_meaning:NTF \\\
2466         { \@@_the_cell_is_empty: }
2467         {
2468             \peek_meaning:NTF \crcr
2469             \@@_the_cell_is_empty:
2470             \group_align_safe_end:
2471         }
2472     }
2473 }
2474 \cs_new_protected:Npn \@@_the_cell_is_empty:
2475 {
2476     \group_align_safe_end:
2477     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2478     {

```

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

```

2479         \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2480         \skip_horizontal:N \l_@@_col_width_dim
2481     }
2482 }
2483 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2484 {
2485     \peek_meaning:NT \__siunitx_table_skip:n
2486     { \bool_gset_true:N \g_@@_empty_cell_bool }
2487 }

```

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.

```

2488 \cs_new_protected:Npn \@@_center_cell_box:
2489 {

```

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

```

2490     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2491     {
2492         \dim_compare:nNnT
2493         { \box_ht:N \l_@@_cell_box }
2494         >

```

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).

```

2495         { \box_ht:N \strutbox }
2496     {
2497         \hbox_set:Nn \l_@@_cell_box
2498         {
2499             \box_move_down:nn
2500             {
2501                 ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2502                   + \baselineskip ) / 2
2503             }
2504             { \box_use:N \l_@@_cell_box }
2505         }
2506     }
2507 }
2508 }

```

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

```

2509 \cs_new_protected:Npn \@@_V: #1 #2
2510 {
2511     \str_if_eq:nnTF { #1 } { [ ] }
2512     { \@@_make_preamble_V_i:w [ ] }
2513     { \@@_make_preamble_V_i:w [ ] { #2 } }
2514 }
2515 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2516 { \@@_make_preamble_V_ii:nn { #1 } }
2517 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2518 {
2519     \str_set:Nn \l_@@_vpos_col_str { p }
2520     \str_set:Nn \l_@@_hpos_col_str { j }
2521     \@@_keys_p_column:n { #1 }
2522     \IfPackageLoadedTF { varwidth }
2523     { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2524     {
2525         \@@_error_or_warning:n { varwidth-not-loaded }
2526         \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2527     }
2528 }

```

For `w` and `W`

```

2529 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2530 \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.

```

2531 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2532 {
2533     \str_if_eq:nnTF { #3 } { s }
2534     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2535     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2536 }

```

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.

```

2537 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2538 {
2539     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2540     \tl_gclear:N \g_@@_pre_cell_tl

```

```

2541 \tl_gput_right:Nn \g_@@_array_preamble_tl
2542 {
2543   > {
2544     \dim_set:Nn \l_@@_col_width_dim { #2 }
2545     \@@_cell_begin:
2546     \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2547   }
2548   c
2549   < {
2550     \@@_cell_end_for_w_s:
2551     #1
2552     \@@_adjust_size_box:
2553     \box_use_drop:N \l_@@_cell_box
2554   }
2555 }
2556 \int_gincr:N \c@jCol
2557 \@@_rec_preamble_after_col:n
2558 }

```

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

```

2559 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2560 {
2561   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2562   \tl_gclear:N \g_@@_pre_cell_tl
2563   \tl_gput_right:Nn \g_@@_array_preamble_tl
2564   {
2565     > {

```

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.

```

2566       \dim_set:Nn \l_@@_col_width_dim { #4 }
2567       \hbox_set:Nw \l_@@_cell_box
2568       \@@_cell_begin:
2569       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2570     }
2571     c
2572     < {
2573       \@@_cell_end:
2574       \hbox_set_end:
2575       #1
2576       \@@_adjust_size_box:
2577       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2578     }
2579   }

```

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

```

2580 \int_gincr:N \c@jCol
2581 \@@_rec_preamble_after_col:n
2582 }

```

```

2583 \cs_new_protected:Npn \@@_special_W:
2584 {
2585   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \l_@@_col_width_dim }
2586   { \@@_warning:n { W~warning } }
2587 }

```

For S (of siunitx).

```

2588 \cs_new_protected:Npn \@@_S: #1 #2
2589 {
2590   \str_if_eq:nnTF { #2 } { [ ] }
2591   { \@@_make_preamble_S:w [ ] }
2592   { \@@_make_preamble_S:w [ ] { #2 } }
2593 }

```

```

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

```

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).

```

2604       \socket_assign_plug:n { nicematrix / siunitx-wrap } { active }
2605       \keys_set:n { siunitx } { #1 }
2606       \@@_cell_begin:
2607       \siunitx_cell_begin:w
2608     }
2609     c
2610     <
2611     {
2612       \siunitx_cell_end:

```

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

```

2613       \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2614       {
2615         \bool_if:NTF \l__siunitx_table_text_bool
2616         { \bool_set_true:N }
2617         { \bool_set_false:N }
2618         \l__siunitx_table_text_bool
2619       }
2620       \@@_cell_end:
2621     }
2622   }

```

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

```

2623       \int_gincr:N \c@jCol
2624       \@@_rec_preamble_after_col:n
2625     }

```

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

```

2626 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2627 {
2628   \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.

```

2629   \int_if_zero:nTF { \c@jCol }
2630   {
2631     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2632     {

```

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

```

2633       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2634       \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2635       \@@_rec_preamble:n #2
2636     }
2637   {
2638     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }

```

```

2639         \@@_make_preamble_iv:nn { #1 } { #2 }
2640     }
2641 }
2642 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2643 }
2644 \cs_set_eq:cc { @@ _ \token_to_str:N [ : ] { @@ _ \token_to_str:N ( : }
2645 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2646 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2647 {
2648     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2649     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2650     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2651     {
2652         \@@_error:nn { delimiter~after~opening } { #2 }
2653         \@@_rec_preamble:n
2654     }
2655     { \@@_rec_preamble:n #2 }
2656 }

```

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

```

2657 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2658 { \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}`).

```

2659 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2660 {
2661     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2662     \tl_if_in:nnTF { ) ] \} } { #2 }
2663     { \@@_make_preamble_v:nnn #1 #2 }
2664     {
2665         \str_if_eq:nnTF { \s_stop } { #2 }
2666         {
2667             \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2668             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2669             {
2670                 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2671                 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2672                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2673                 \@@_rec_preamble:n #2
2674             }
2675         }
2676         {
2677             \tl_if_in:nnT { ( [ \{ \left } { #2 }
2678             { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2679             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2680             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2681             \@@_rec_preamble:n #2
2682         }
2683     }
2684 }
2685 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2686 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2687 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2688 {
2689     \str_if_eq:nnTF { \s_stop } { #3 }
2690     {
2691         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2692         {
2693             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }

```

```

2694 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2695 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2696 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2697 }
2698 {
2699 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2700 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2701 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2702 \@@_error:nn { double~closing~delimiter } { #2 }
2703 }
2704 }
2705 {
2706 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2707 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2708 \@@_error:nn { double~closing~delimiter } { #2 }
2709 \@@_rec_preamble:n #3
2710 }
2711 }

2712 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2713 { \use:c { @@ _ \token_to_str:N ) : } }

```

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

```

2714 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2715 {
2716 \str_if_eq:nnTF { #1 } { < }
2717 { \@@_rec_preamble_after_col_i:n }
2718 {
2719 \str_if_eq:nnTF { #1 } { @ }
2720 { \@@_rec_preamble_after_col_ii:n }
2721 {
2722 \str_if_eq:eeTF \l_@@_vlines_clist { all }
2723 {
2724 \tl_gput_right:Nn \g_@@_array_preamble_tl
2725 { ! { \skip_horizontal:N \arrayrulewidth } }
2726 }
2727 {
2728 \clist_if_in:NiT \l_@@_vlines_clist
2729 { \int_eval:n { \c@jCol + 1 } }
2730 {
2731 \tl_gput_right:Nn \g_@@_array_preamble_tl
2732 { ! { \skip_horizontal:N \arrayrulewidth } }
2733 }
2734 }
2735 \@@_rec_preamble:n { #1 }
2736 }
2737 }
2738 }

2739 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2740 {
2741 \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2742 \@@_rec_preamble_after_col:n
2743 }

```

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

```

2744 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2745 {
2746 \str_if_eq:eeTF \l_@@_vlines_clist { all }
2747 {

```



```

2748 \tl_gput_right:Nn \g_@@_array_preamble_tl
2749 { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2750 }
2751 {
2752 \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2753 {
2754 \tl_gput_right:Nn \g_@@_array_preamble_tl
2755 { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2756 }
2757 { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2758 }
2759 \@@_rec_preamble:n
2760 }

2761 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2762 {
2763 \tl_clear:N \l_tmpa_tl
2764 \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2765 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2766 }

```

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.

```

2767 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2768 { \@@_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.

```

2769 \cs_new_protected:Npn \@@_X: #1 #2
2770 {
2771 \str_if_eq:nnTF { #2 } { [ ]
2772 { \@@_make_preamble_X:w [ ]
2773 { \@@_make_preamble_X:w [ ] #2 }
2774 }
2775 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2776 { \@@_make_preamble_X:i:n { #1 } }

```

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

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

```

2777 \keys_define:nn { nicematrix / X-column }
2778 {
2779 unknown .code:n =
2780 \regex_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2781 { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2782 { \@@_error_or_warning:n { invalid-weight } }
2783 }

```

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

```

2784 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2785 {

```

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).

```

2786 \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).

```

2787 \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`).

```
2788 \fp_set:Nn \l_tmpa_fp { 1.0 }
2789 \@@_keys_p_column:n { #1 }
```

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

```
2790 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
```

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

```
2791 \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).

```
2792 \bool_if:NTF \l_@@_X_columns_aux_bool
2793 {
2794 \@@_make_preamble_ii_iv:nnn
```

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

```
2795 { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2796 { minipage }
2797 { \@@_no_update_width: }
2798 }
```

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.

```
2799 {
2800 \tl_gput_right:Nn \g_@@_array_preamble_tl
2801 {
2802 > {
2803 \@@_cell_begin:
2804 \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`.

```
2805 \NotEmpty
```

The following code will nullify the box of the cell.

```
2806 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2807 { \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`.

```
2808 \begin { minipage } { 5 cm } \arraybackslash
2809 }
2810 c
2811 < {
2812 \end { minipage }
2813 \@@_cell_end:
2814 }
2815 }
2816 \int_gincr:N \c@jCol
2817 \@@_rec_preamble_after_col:n
2818 }
2819 }
```

```
2820 \cs_new_protected:Npn \@@_no_update_width:
2821 {
2822 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2823 { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2824 }
```

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

```

2825 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2826 {
2827   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2828   { \int_eval:n { \c@jCol + 1 } }
2829   \tl_gput_right:Ne \g_@@_array_preamble_tl
2830   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2831   \@@_rec_preamble:n
2832 }

```

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.

```

2833 \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).

```

2834 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2835 { \@@_fatal:n { Preamble-forgotten } }
2836 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
2837 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
2838 { @@ _ \token_to_str:N \hline : }
2839 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
2840 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
2841 { @@ _ \token_to_str:N \hline : }
2842 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
2843 { @@ _ \token_to_str:N \hline : }
2844 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
2845 { @@ _ \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).

```

2846 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2847 {

```

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`.

```

2848   \multispan { #1 }
2849   \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2850   \begingroup
2851   \bool_if:NT \c_@@_testphase_table_bool
2852   { \tbl_update_multicolumn_cell_data:n { #1 } }
2853   \def \@@addamp
2854   { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2855   \tl_gclear:N \g_@@_preamble_tl
2856   \@@_make_m_preamble:n #2 \q_stop

```

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

```

2857   \exp_args:No \@mkpream \g_@@_preamble_tl
2858   \@addtopreamble \@empty
2859   \endgroup
2860   \bool_if:NT \c_@@_recent_array_bool
2861   { \UseTaggingSocket { tbl / colspan } { #1 } }

```

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

```

2862 \int_compare:nNtT { #1 } > { \c_one_int }
2863 {
2864   \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2865   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2866   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2867   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2868   {
2869     {
2870       \int_if_zero:nTF { \c@jCol }
2871       { \int_eval:n { \c@iRow + 1 } }
2872       { \int_use:N \c@iRow }
2873     }
2874     { \int_eval:n { \c@jCol + 1 } }
2875     {
2876       \int_if_zero:nTF { \c@jCol }
2877       { \int_eval:n { \c@iRow + 1 } }
2878       { \int_use:N \c@iRow }
2879     }
2880     { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block

```

2881     { }
2882   }
2883 }

```

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

```

2884 \RenewDocumentCommand \cellcolor { 0 { } m }
2885 {
2886   \tl_gput_right:Ne \g_@@_pre_code_before_tl
2887   {
2888     \@@_rectanglecolor [ ##1 ]
2889     { \exp_not:n { ##2 } }
2890     { \int_use:N \c@iRow - \int_use:N \c@jCol }
2891     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2892   }
2893   \ignorespaces
2894 }

```

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

```

2895 \def \@sharp { #3 }
2896 \@arstrut
2897 \@preamble
2898 \null

```

We add some lines.

```

2899 \int_gadd:Nn \c@jCol { #1 - 1 }
2900 \int_compare:nNtT { \c@jCol } > { \g_@@_col_total_int }
2901 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2902 \ignorespaces
2903 }

```

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`.

```

2904 \cs_new_protected:Npn \@_make_m_preamble:n #1
2905 {
2906   \str_case:nnF { #1 }
2907   {
2908     c { \@_make_m_preamble_i:n #1 }
2909     l { \@_make_m_preamble_i:n #1 }

```

```

2910     r { \@@_make_m_preamble_i:n #1 }
2911     > { \@@_make_m_preamble_ii:nn #1 }
2912     ! { \@@_make_m_preamble_ii:nn #1 }
2913     @ { \@@_make_m_preamble_ii:nn #1 }
2914     | { \@@_make_m_preamble_iii:n #1 }
2915     p { \@@_make_m_preamble_iv:nnn t #1 }
2916     m { \@@_make_m_preamble_iv:nnn c #1 }
2917     b { \@@_make_m_preamble_iv:nnn b #1 }
2918     w { \@@_make_m_preamble_v:nnnn { } #1 }
2919     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2920     \q_stop { }
2921   }
2922   {
2923     \cs_if_exist:cTF { NC @ find @ #1 }
2924     {
2925       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2926       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2927     }
2928     {
2929       \str_if_eq:nnTF { #1 } { S }
2930       { \@@_fatal:n { unknown~column~type~S } }
2931       { \@@_fatal:nn { unknown~column~type } { #1 } }
2932     }
2933   }
2934 }

```

For c, l and r

```

2935 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2936 {
2937   \tl_gput_right:Nn \g_@@_preamble_tl
2938   {
2939     > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
2940     #1
2941     < \@@_cell_end:
2942   }

```

We test for the presence of a <.

```

2943   \@@_make_m_preamble_x:n
2944 }

```

For >, ! and @

```

2945 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2946 {
2947   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2948   \@@_make_m_preamble:n
2949 }

```

For |

```

2950 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2951 {
2952   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2953   \@@_make_m_preamble:n
2954 }

```

For p, m and b

```

2955 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2956 {
2957   \tl_gput_right:Nn \g_@@_preamble_tl
2958   {
2959     > {
2960       \@@_cell_begin:
2961       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2962       \mode_leave_vertical:

```

```

2963         \arraybackslash
2964         \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2965     }
2966     c
2967     < {
2968         \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2969         \end { minipage }
2970         \@@_cell_end:
2971     }
2972 }

```

We test for the presence of a <.

```

2973     \@@_make_m_preamble_x:n
2974 }

```

For w and W

```

2975 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2976 {
2977     \tl_gput_right:Nn \g_@@_preamble_tl
2978     {
2979         > {
2980             \dim_set:Nn \l_@@_col_width_dim { #4 }
2981             \hbox_set:Nw \l_@@_cell_box
2982             \@@_cell_begin:
2983             \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2984         }
2985         c
2986         < {
2987             \@@_cell_end:
2988             \hbox_set_end:
2989             \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
2990             #1
2991             \@@_adjust_size_box:
2992             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2993         }
2994     }

```

We test for the presence of a <.

```

2995     \@@_make_m_preamble_x:n
2996 }

```

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

```

2997 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2998 {
2999     \str_if_eq:nnTF { #1 } { < }
3000     { \@@_make_m_preamble_ix:n }
3001     { \@@_make_m_preamble:n { #1 } }
3002 }
3003 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3004 {
3005     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3006     \@@_make_m_preamble_x:n
3007 }

```

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).

```

3008 \cs_new_protected:Npn \@@_put_box_in_flow:
3009 {
3010     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3011     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3012     \str_if_eq:eeTF \l_@@_baseline_tl { c }

```

```

3013 { \box_use_drop:N \l_tmpa_box }
3014 { \@@_put_box_in_flow_i: }
3015 }

```

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

```

3016 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3017 {
3018   \pgfpicture
3019     \@@_qpoint:n { row - 1 }
3020     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3021     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3022     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3023     \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).

```

3024   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3025   {
3026     \int_set:Nn \l_tmpa_int
3027     {
3028       \str_range:Nnn
3029         \l_@@_baseline_tl
3030         6
3031         { \tl_count:o \l_@@_baseline_tl }
3032     }
3033     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3034   }
3035   {
3036     \str_if_eq:eeTF \l_@@_baseline_tl { t }
3037     { \int_set_eq:NN \l_tmpa_int \c_one_int }
3038     {
3039       \str_if_eq:onTF \l_@@_baseline_tl { b }
3040       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3041       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3042     }
3043     \bool_lazy_or:nnT
3044     { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3045     { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3046     {
3047       \@@_error:n { bad-value-for-baseline }
3048       \int_set_eq:NN \l_tmpa_int \c_one_int
3049     }
3050     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3051     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3052   }
3053   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3054   \endpgfpicture
3055   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3056   \box_use_drop:N \l_tmpa_box
3057 }

```

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).

```

3058 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3059 {

```

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.

```

3060 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3061 {
3062   \int_compare:nNnT { \c@jCol } > { \c_one_int }
3063   {
3064     \box_set_wd:Nn \l_@@_the_array_box
3065     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3066   }
3067 }

```

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

```

3068 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3069 \bool_if:NT \l_@@_caption_above_bool
3070 {
3071   \tl_if_empty:NF \l_@@_caption_tl
3072   {
3073     \bool_set_false:N \g_@@_caption_finished_bool
3074     \int_gzero:N \c@tabularnote
3075     \@@_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).

```

3076   \int_compare:nNnT { \g_@@_notes_caption_int } > { \c_zero_int }
3077   {
3078     \tl_gput_right:Ne \g_@@_aux_tl
3079     {
3080       \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3081       { \int_use:N \g_@@_notes_caption_int }
3082     }
3083     \int_gzero:N \g_@@_notes_caption_int
3084   }
3085 }
3086 }

```

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

```

3087 \hbox
3088 {
3089   \box_use_drop:N \l_@@_the_array_box

```

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

```

3090 \@@_create_extra_nodes:
3091 \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3092 }

```

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).

```

3093 \bool_lazy_any:nT
3094 {
3095   { ! \seq_if_empty_p:N \g_@@_notes_seq }
3096   { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3097   { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3098 }
3099 \@@_insert_tabularnotes:
3100 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3101 \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }

```



```

3102 \end { minipage }
3103 }

```

```

3104 \cs_new_protected:Npn \@@_insert_caption:
3105 {
3106   \tl_if_empty:NF \l_@@_caption_tl
3107   {
3108     \cs_if_exist:NTF \@capttype
3109     { \@@_insert_caption_i: }
3110     { \@@_error:n { caption~outside~float } }
3111   }
3112 }

```

```

3113 \cs_new_protected:Npn \@@_insert_caption_i:
3114 {
3115   \group_begin:

```

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

```

3116   \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.

```

3117   \IfPackageLoadedT { floatrow }
3118   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3119   \tl_if_empty:NTF \l_@@_short_caption_tl
3120   { \caption }
3121   { \caption [ \l_@@_short_caption_tl ] }
3122   { \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.

```

3123   \bool_if:NF \g_@@_caption_finished_bool
3124   {
3125     \bool_gset_true:N \g_@@_caption_finished_bool
3126     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3127     \int_gzero:N \c@tabularnote
3128   }
3129   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3130   \group_end:
3131 }

```

```

3132 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3133 {
3134   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3135   \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3136 }
3137 \cs_new_protected:Npn \@@_insert_tabularnotes:
3138 {
3139   \seq_concat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3140   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3141   \skip_vertical:N 0.65ex

```

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

```

3142   \group_begin:
3143   \l_@@_notes_code_before_tl
3144   \tl_if_empty:NF \g_@@_tabularnote_tl
3145   {
3146     \g_@@_tabularnote_tl \par

```

```

3147     \tl_gclear:N \g_@@_tabularnote_tl
3148 }

```

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.

```

3149 \int_compare:nNnT { \c@tabularnote } > { \c_zero_int }
3150 {
3151     \bool_if:NTF \l_@@_notes_para_bool
3152     {
3153         \begin { tabularnotes* }
3154         \seq_map_inline:Nn \g_@@_notes_seq
3155             { \@@_one_tabularnote:nn ##1 }
3156         \strut
3157         \end { tabularnotes* }

```

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

```

3158     \par
3159 }
3160 {
3161     \tabularnotes
3162     \seq_map_inline:Nn \g_@@_notes_seq
3163         { \@@_one_tabularnote:nn ##1 }
3164     \strut
3165     \endtabularnotes
3166 }
3167 }
3168 \unskip
3169 \group_end:
3170 \bool_if:NT \l_@@_notes_bottomrule_bool
3171 {
3172     \IfPackageLoadedTF { booktabs }
3173     {

```

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

```

3174     \skip_vertical:N \aboverulesep

```

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

```

3175     { \CT@arc@ \hrule height \heavyrulewidth }
3176 }
3177 { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3178 }
3179 \l_@@_notes_code_after_tl
3180 \seq_gclear:N \g_@@_notes_seq
3181 \seq_gclear:N \g_@@_notes_in_caption_seq
3182 \int_gzero:N \c@tabularnote
3183 }

```

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.

```

3184 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3185 {
3186     \tl_if_novalue:nTF { #1 }
3187     { \item }
3188     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3189 }

```

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`.

```

3190 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3191 {

```

```

3192 \pgfpicture
3193   \@@_qpoint:n { row - 1 }
3194   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3195   \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3196   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3197 \endpgfpicture
3198 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3199 \int_if_zero:nT { \l_@@_first_row_int }
3200 {
3201   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3202   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3203 }
3204 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3205 }

```

Now, the general case.

```

3206 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3207 {

```

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

```

3208   \str_if_eq:eeT \l_@@_baseline_tl { t }
3209   { \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`.

```

3210 \pgfpicture
3211 \@@_qpoint:n { row - 1 }
3212 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3213 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3214 {
3215   \int_set:Nn \l_tmpa_int
3216   {
3217     \str_range:Nnn
3218       \l_@@_baseline_tl
3219       { 6 }
3220     { \tl_count:o \l_@@_baseline_tl }
3221   }
3222   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3223 }
3224 {
3225   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3226   \bool_lazy_or:nnT
3227     { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3228     { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3229   {
3230     \@@_error:n { bad~value~for~baseline }
3231     \int_set:Nn \l_tmpa_int 1
3232   }
3233   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3234 }
3235 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3236 \endpgfpicture
3237 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3238 \int_if_zero:nT { \l_@@_first_row_int }
3239 {
3240   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3241   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3242 }
3243 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3244 }

```

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.

```

3245 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3246 {

```

We will compute the real width of both delimiters used.

```

3247 \dim_zero_new:N \l_@@_real_left_delim_dim
3248 \dim_zero_new:N \l_@@_real_right_delim_dim
3249 \hbox_set:Nn \l_tmpb_box
3250 {
3251 \m@th % added 2024/11/21
3252 \c_math_toggle_token
3253 \left #1
3254 \vcenter
3255 {
3256 \vbox_to_ht:nn
3257 { \box_ht_plus_dp:N \l_tmpa_box }
3258 { }
3259 }
3260 \right .
3261 \c_math_toggle_token
3262 }
3263 \dim_set:Nn \l_@@_real_left_delim_dim
3264 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3265 \hbox_set:Nn \l_tmpb_box
3266 {
3267 \m@th % added 2024/11/21
3268 \c_math_toggle_token
3269 \left .
3270 \vbox_to_ht:nn
3271 { \box_ht_plus_dp:N \l_tmpa_box }
3272 { }
3273 \right #2
3274 \c_math_toggle_token
3275 }
3276 \dim_set:Nn \l_@@_real_right_delim_dim
3277 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3278 \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3279 \@@_put_box_in_flow:
3280 \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3281 }

```

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).

```

3282 \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).

```

3283 {
3284 \peek_remove_spaces:n
3285 {
3286 \peek_meaning:NTF \end
3287 { \@@_analyze_end:Nn }
3288 {
3289 \@@_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`).

```

3290 \@@_array:o \g_@@_array_preamble_tl

```

```

3291     }
3292   }
3293 }
3294 {
3295   \@@_create_col_nodes:
3296   \endarray
3297 }

```

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

```

3298 \NewDocumentEnvironment { @@-light-syntax } { b }
3299 {

```

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`.

```

3300   \tl_if_empty:nT { #1 }
3301   { \@@_fatal:n { empty~environment } }
3302   \tl_if_in:nnT { #1 } { & }
3303   { \@@_fatal:n { ampersand~in~light-syntax } }
3304   \tl_if_in:nnT { #1 } { \ }
3305   { \@@_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`.

```

3306   \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3307 }

```

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.

```

3308 {
3309   \@@_create_col_nodes:
3310   \endarray
3311 }
3312 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3313 {
3314   \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).

```

3315   \seq_clear_new:N \l_@@_rows_seq

```

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

```

3316   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3317   \bool_if:NTF \l_@@_light_syntax_expanded_bool
3318   { \seq_set_split:Nee }
3319   { \seq_set_split:Nnn }
3320   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3321   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3322   \tl_if_empty:NF \l_tmpa_tl
3323   { \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.

```

3324   \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
3325   { \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`).

```
3326 \tl_build_begin:N \l_@@_new_body_tl
3327 \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3328 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3329 \@@_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).

```
3330 \seq_map_inline:Nn \l_@@_rows_seq
3331 {
3332   \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3333   \@@_line_with_light_syntax:n { #1 }
3334 }
3335 \tl_build_end:N \l_@@_new_body_tl
3336 \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
3337 {
3338   \int_set:Nn \l_@@_last_col_int
3339     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3340 }
```

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.

```
3341 \@@_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`).

```
3342 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3343 }
3344 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3345 {
3346   \seq_clear_new:N \l_@@_cells_seq
3347   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3348   \int_set:Nn \l_@@_nb_cols_int
3349     {
3350       \int_max:nn
3351         { \l_@@_nb_cols_int }
3352         { \seq_count:N \l_@@_cells_seq }
3353     }
3354   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3355   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3356   \seq_map_inline:Nn \l_@@_cells_seq
3357     { \tl_build_put_right:Nn \l_@@_new_body_tl { & #1 } }
3358 }
3359 \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`.

```
3360 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3361 {
3362   \str_if_eq:eeT \g_@@_name_env_str { #2 }
3363     { \@@_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.

```
3364 \end { #2 }
3365 }
```

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`).

```

3366 \cs_new:Npn \@@_create_col_nodes:
3367 {
3368   \crrc
3369   \int_if_zero:nT { \l_@@_first_col_int }
3370   {
3371     \omit
3372     \hbox_overlap_left:n
3373     {
3374       \bool_if:NT \l_@@_code_before_bool
3375       { \pgfsys@markposition { \@@_env: - col - 0 } }
3376       \pgfpicture
3377       \pgfrememberpicturepositiononpagetrue
3378       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3379       \str_if_empty:NF \l_@@_name_str
3380       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3381       \endpgfpicture
3382       \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3383     }
3384     &
3385   }
3386   \omit

```

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

```

3387   \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`).

```

3388   \int_if_zero:nTF { \l_@@_first_col_int }
3389   {
3390     \@@_mark_position:n { 1 }
3391     \pgfpicture
3392     \pgfrememberpicturepositiononpagetrue
3393     \pgfcoordinate { \@@_env: - col - 1 }
3394     { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
3395     \str_if_empty:NF \l_@@_name_str
3396     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3397     \endpgfpicture
3398   }
3399   {
3400     \bool_if:NT \l_@@_code_before_bool
3401     {
3402       \hbox
3403       {
3404         \skip_horizontal:n { 0.5 \arrayrulewidth }
3405         \pgfsys@markposition { \@@_env: - col - 1 }
3406         \skip_horizontal:n { -0.5 \arrayrulewidth }
3407       }
3408     }
3409     \pgfpicture
3410     \pgfrememberpicturepositiononpagetrue
3411     \pgfcoordinate { \@@_env: - col - 1 }
3412     { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }
3413     \@@_node_alias:n { 1 }
3414     \endpgfpicture
3415   }

```

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.

```

3416 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3417 \bool_if:NF \l_@@_auto_columns_width_bool
3418 { \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim } }
3419 {
3420   \bool_lazy_and:nnTF
3421     { \l_@@_auto_columns_width_bool }
3422     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3423     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3424     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3425   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3426 }
3427 \skip_horizontal:N \g_tmpa_skip
3428 \hbox
3429 {
3430   \@@_mark_position:n { 2 }
3431   \pgfpicture
3432   \pgfrememberpicturepositiononpagetrue
3433   \pgfcoordinate { \@@_env: - col - 2 }
3434   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3435   \@@_node_alias:n { 2 }
3436   \endpgfpicture
3437 }

```

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.

```

3438 \int_gset_eq:NN \g_tmpa_int \c_one_int
3439 \bool_if:NTF \g_@@_last_col_found_bool
3440 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3441 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3442 {
3443   &
3444   \omit
3445   \int_gincr:N \g_tmpa_int

```

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

```

3446 \skip_horizontal:N \g_tmpa_skip
3447 \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

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

```

3448 \pgfpicture
3449 \pgfrememberpicturepositiononpagetrue
3450 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3451 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3452 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3453 \endpgfpicture
3454 }

3455 &
3456 \omit

```

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

```

3457 \bool_lazy_or:nnF
3458 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3459 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3460 {
3461   \skip_horizontal:N \g_tmpa_skip
3462   \int_gincr:N \g_tmpa_int
3463   \bool_lazy_any:nF
3464   {
3465     \g_@@_delims_bool
3466     \l_@@_tabular_bool

```



```

3467         { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3468         \l_@@_exterior_arraycolsep_bool
3469         \l_@@_bar_at_end_of_pream_bool
3470     }
3471     { \skip_horizontal:n { - \col@sep } }
3472 \bool_if:NT \l_@@_code_before_bool
3473     {
3474         \hbox
3475         {
3476             \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.

```

3477         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3478         { \skip_horizontal:n { - \arraycolsep } }
3479         \pgfsys@markposition
3480         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3481         \skip_horizontal:n { 0.5 \arrayrulewidth }
3482         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3483         { \skip_horizontal:N \arraycolsep }
3484     }
3485 }
3486 \pgfpicture
3487 \pgfrememberpicturepositiononpagetrue
3488 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3489 {
3490     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3491     {
3492         \pgfpoint
3493         { - 0.5 \arrayrulewidth - \arraycolsep }
3494         \c_zero_dim
3495     }
3496     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3497 }
3498 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3499 \endpgfpicture
3500 }

3501 \bool_if:NT \g_@@_last_col_found_bool
3502 {
3503     \hbox_overlap_right:n
3504     {
3505         \skip_horizontal:N \g_@@_width_last_col_dim
3506         \skip_horizontal:N \col@sep
3507         \bool_if:NT \l_@@_code_before_bool
3508         {
3509             \pgfsys@markposition
3510             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3511         }
3512         \pgfpicture
3513         \pgfrememberpicturepositiononpagetrue
3514         \pgfcoordinate
3515         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3516         \pgfpointorigin
3517         \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3518         \endpgfpicture
3519     }
3520 }
3521 % \cr
3522 }

```

```

3523 \cs_new_protected:Npn \@@_mark_position:n #1
3524 {
3525   \bool_if:NT \l_@@_code_before_bool
3526   {
3527     \hbox
3528     {
3529       \skip_horizontal:n { -0.5 \arrayrulewidth }
3530       \pgfsys@markposition { \@@_env: - col - #1 }
3531       \skip_horizontal:n { 0.5 \arrayrulewidth }
3532     }
3533   }
3534 }
3535 \cs_new_protected:Npn \@@_node_alias:n #1
3536 {
3537   \str_if_empty:NF \l_@@_name_str
3538   { \pgfnodelalias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3539 }

```

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

```

3540 \tl_const:Nn \c_@@_preamble_first_col_tl
3541 {
3542   >
3543   {

```

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

```

3544   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3545   \bool_gset_true:N \g_@@_after_col_zero_bool
3546   \@@_begin_of_row:
3547   \hbox_set:Nw \l_@@_cell_box
3548   \@@_math_toggle:
3549   \@@_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”.

```

3550   \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3551   {
3552     \bool_lazy_or:nnT
3553     { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3554     { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3555     {
3556       \l_@@_code_for_first_col_tl
3557       \xglobal \colorlet { nicematrix-first-col } { . }
3558     }
3559   }
3560 }

```

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`.

```

3561   l
3562   <
3563   {
3564     \@@_math_toggle:
3565     \hbox_set_end:
3566     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3567     \@@_adjust_size_box:
3568     \@@_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.

```

3569   \dim_gset:Nn \g_@@_width_first_col_dim
3570   { \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.

```

3571 \hbox_overlap_left:n
3572 {
3573   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3574   { \@@_node_cell: }
3575   { \box_use_drop:N \l_@@_cell_box }
3576   \skip_horizontal:N \l_@@_left_delim_dim
3577   \skip_horizontal:N \l_@@_left_margin_dim
3578   \skip_horizontal:N \l_@@_extra_left_margin_dim
3579 }
3580 \bool_gset_false:N \g_@@_empty_cell_bool
3581 \skip_horizontal:n { -2 \col@sep }
3582 }
3583 }

```

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

```

3584 \tl_const:Nn \c_@@_preamble_last_col_tl
3585 {
3586   >
3587   {
3588     \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).

```

3589 \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.

```

3590 \bool_gset_true:N \g_@@_last_col_found_bool
3591 \int_gincr:N \c@jCol
3592 \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3593 \hbox_set:Nw \l_@@_cell_box
3594 \@@_math_toggle:
3595 \@@_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”.

```

3596 \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3597 {
3598   \bool_lazy_or:nnT
3599   { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3600   { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3601   {
3602     \l_@@_code_for_last_col_tl
3603     \xglobal \colorlet { nicematrix-last-col } { . }
3604   }
3605 }
3606 }
3607 1
3608 <
3609 {
3610   \@@_math_toggle:
3611   \hbox_set_end:
3612   \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3613   \@@_adjust_size_box:
3614   \@@_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.

```

3615 \dim_gset:Nn \g_@@_width_last_col_dim
3616 { \dim_max:nn { \g_@@_width_last_col_dim } { \box_wd:N \l_@@_cell_box } }
3617 \skip_horizontal:n { -2 \col@sep }

```

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

```

3618 \hbox_overlap_right:n
3619 {

```

```

3620         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3621         {
3622             \skip_horizontal:N \l_@@_right_delim_dim
3623             \skip_horizontal:N \l_@@_right_margin_dim
3624             \skip_horizontal:N \l_@@_extra_right_margin_dim
3625             \@@_node_cell:
3626         }
3627     }
3628     \bool_gset_false:N \g_@@_empty_cell_bool
3629 }
3630 }

```

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

```

3631 \NewDocumentEnvironment { NiceArray } { }
3632 {
3633     \bool_gset_false:N \g_@@_delims_bool
3634     \str_if_empty:NT \g_@@_name_env_str
3635     { \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).

```

3636     \NiceArrayWithDelims . .
3637 }
3638 { \endNiceArrayWithDelims }

```

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

```

3639 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3640 {
3641     \NewDocumentEnvironment { #1 NiceArray } { }
3642     {
3643         \bool_gset_true:N \g_@@_delims_bool
3644         \str_if_empty:NT \g_@@_name_env_str
3645         { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3646         \@@_test_if_math_mode:
3647         \NiceArrayWithDelims #2 #3
3648     }
3649     { \endNiceArrayWithDelims }
3650 }
3651 \@@_def_env:NNN p ( )
3652 \@@_def_env:NNN b [ ]
3653 \@@_def_env:NNN B \{ \}
3654 \@@_def_env:NNN v \vert \vert
3655 \@@_def_env:NNN V \Vert \Vert

```

13 The environment `{NiceMatrix}` and its variants

```

3656 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3657 {
3658     \bool_set_false:N \l_@@_preamble_bool
3659     \tl_clear:N \l_tmpa_tl
3660     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3661     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3662     \tl_put_right:Nn \l_tmpa_tl
3663     {
3664         *
3665         {

```

```

3666         \int_case:nnF \l_@@_last_col_int
3667         {
3668             { -2 } { \c@MaxMatrixCols }
3669             { -1 } { \int_eval:n { \c@MaxMatrixCols + 1 } }
3670         }
3671         { \int_eval:n { \l_@@_last_col_int - 1 } }
3672     }
3673     { #2 }
3674 }
3675 \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3676 \exp_args:No \l_tmpb_tl \l_tmpa_tl
3677 }
3678 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3679 \clist_map_inline:nn { p , b , B , v , V }
3680 {
3681     \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3682     {
3683         \bool_gset_true:N \g_@@_delims_bool
3684         \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3685         \int_if_zero:nT { \l_@@_last_col_int }
3686         {
3687             \bool_set_true:N \l_@@_last_col_without_value_bool
3688             \int_set:Nn \l_@@_last_col_int { -1 }
3689         }
3690         \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3691         \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3692     }
3693     { \use:c { end #1 NiceArray } }
3694 }

```

We define also an environment {NiceMatrix}

```

3695 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3696 {
3697     \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3698     \int_if_zero:nT { \l_@@_last_col_int }
3699     {
3700         \bool_set_true:N \l_@@_last_col_without_value_bool
3701         \int_set:Nn \l_@@_last_col_int { -1 }
3702     }
3703     \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3704     \bool_lazy_or:nnT
3705     { \clist_if_empty_p:N \l_@@_vlines_clist }
3706     { \l_@@_except_borders_bool }
3707     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3708     \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3709 }
3710 { \endNiceArray }

```

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

```

3711 \cs_new_protected:Npn \@@_NotEmpty:
3712 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3713 \NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
3714 {

```

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.

```

3715 \dim_compare:nNnT { \l_@@_width_dim } = { \c_zero_dim }
3716 { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3717 \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3718 \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3719 \tl_if_empty:NF \l_@@_short_caption_tl
3720 {
3721   \tl_if_empty:NT \l_@@_caption_tl
3722   {
3723     \@@_error_or_warning:n { short-caption-without~caption }
3724     \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3725   }
3726 }
3727 \tl_if_empty:NF \l_@@_label_tl
3728 {
3729   \tl_if_empty:NT \l_@@_caption_tl
3730   { \@@_error_or_warning:n { label~without~caption } }
3731 }
3732 \NewDocumentEnvironment { TabularNote } { b }
3733 {
3734   \bool_if:NTF \l_@@_in_code_after_bool
3735   { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3736   {
3737     \tl_if_empty:NF \g_@@_tabularnote_tl
3738     { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3739     \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3740   }
3741 }
3742 { }
3743 \@@_settings_for_tabular:
3744 \NiceArray { #2 }
3745 }
3746 { \endNiceArray }
3747 \cs_new_protected:Npn \@@_settings_for_tabular:
3748 {
3749   \bool_set_true:N \l_@@_tabular_bool
3750   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3751   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3752   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3753 }

3754 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3755 {
3756   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3757   \dim_set:Nn \l_@@_width_dim { #1 }
3758   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3759   \@@_settings_for_tabular:
3760   \NiceArray { #3 }
3761 }
3762 {
3763   \endNiceArray
3764   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3765   { \@@_error:n { NiceTabularX~without~X } }
3766 }

3767 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3768 {
3769   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3770   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3771   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3772   \@@_settings_for_tabular:
3773   \NiceArray { #3 }
3774 }
3775 { \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`).

```

3776 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3777 {
3778   \bool_lazy_all:nT
3779   {
3780     { \dim_compare_p:nNn { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim } }
3781     { \l_@@_hvlines_bool }
3782     { ! \g_@@_delims_bool }
3783     { ! \l_@@_except_borders_bool }
3784   }
3785   {
3786     \bool_set_true:N \l_@@_except_borders_bool
3787     \clist_if_empty:NF \l_@@_corners_clist
3788     { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3789     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3790     {
3791       \@@_stroke_block:nnn
3792       {
3793         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3794         draw = \l_@@_rules_color_tl
3795       }
3796       { 1-1 }
3797       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3798     }
3799   }
3800 }

3801 \cs_new_protected:Npn \@@_after_array:
3802 {

```

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

```

3803   \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3804   \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.

```

3805   \bool_if:NT \g_@@_last_col_found_bool
3806   { \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`.

```

3807   \bool_if:NT \l_@@_last_col_without_value_bool
3808   { \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.

```

3809   \bool_if:NT \l_@@_last_row_without_value_bool
3810   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3811 \tl_gput_right:Ne \g_@@_aux_tl
3812 {
3813   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3814   {
3815     \int_use:N \l_@@_first_row_int ,
3816     \int_use:N \c@iRow ,
3817     \int_use:N \g_@@_row_total_int ,
3818     \int_use:N \l_@@_first_col_int ,
3819     \int_use:N \c@jCol ,
3820     \int_use:N \g_@@_col_total_int
3821   }
3822 }

```

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`).

```

3823 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3824 {
3825   \tl_gput_right:Ne \g_@@_aux_tl
3826   {
3827     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3828     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3829   }
3830 }
3831 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3832 {
3833   \tl_gput_right:Ne \g_@@_aux_tl
3834   {
3835     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3836     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3837     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3838     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3839   }
3840 }

```

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

```

3841 \@@_create_diag_nodes:

```

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

```

3842 \pgfpicture
3843 \@@_create_aliases_last:
3844 \str_if_empty:NF \l_@@_name_str { \@@_create_alias_nodes: }
3845 \endpgfpicture

```

By default, the diagonal lines will be parallelized¹². 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.

```

3846 \bool_if:NT \l_@@_parallelize_diags_bool
3847 {
3848   \int_gzero:N \g_@@_ddots_int
3849   \int_gzero:N \g_@@_iddots_int

```

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the Δ_x and Δ_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 Δ_x and Δ_y of the first `\Iddots` diagonal.

```

3850   \dim_gzero:N \g_@@_delta_x_one_dim
3851   \dim_gzero:N \g_@@_delta_y_one_dim
3852   \dim_gzero:N \g_@@_delta_x_two_dim
3853   \dim_gzero:N \g_@@_delta_y_two_dim
3854 }

```

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


```

3855 \bool_set_false:N \l_@@_initial_open_bool
3856 \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.

```

3857 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

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

```

3858 \@@_draw_dotted_lines:

```

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

```

3859 \clist_if_empty:NF \l_@@_corners_clist
3860 {
3861   \bool_if:NTF \l_@@_no_cell_nodes_bool
3862     { \@@_error:n { corners-with-no-cell-nodes } }
3863     { \@@_compute_corners: }
3864 }

```

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

```

3865 \@@_adjust_pos_of_blocks_seq:
3866 \@@_deal_with_rounded_corners:
3867 \clist_if_empty:NF \l_@@_hlines_clist { \@@_draw_hlines: }
3868 \clist_if_empty:NF \l_@@_vlines_clist { \@@_draw_vlines: }

```

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

```

3869 \IfPackageLoadedT { tikz }
3870 {
3871   \tikzset
3872   {
3873     every~picture / .style =
3874     {
3875       overlay ,
3876       remember~picture ,
3877       name~prefix = \@@_env: -
3878     }
3879   }
3880 }
3881 \bool_if:NT \c_@@_recent_array_bool
3882 { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3883 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3884 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3885 \cs_set_eq:NN \OverBrace \@@_OverBrace
3886 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3887 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3888 \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.

```

3889 \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
3890 \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.

```

3891 \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).

```
3892 \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`).

```
3893 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
3894 { \@@_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:`.

```
3895 \bool_set_true:N \l_@@_in_code_after_bool
3896 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3897 \scan_stop:
3898 \tl_gclear:N \g_nicematrix_code_after_tl
3899 \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.

```
3900 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3901 \tl_if_empty:NF \g_@@_pre_code_before_tl
3902 {
3903   \tl_gput_right:Ne \g_@@_aux_tl
3904   {
3905     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3906     { \exp_not:o \g_@@_pre_code_before_tl }
3907   }
3908   \tl_gclear:N \g_@@_pre_code_before_tl
3909 }
3910 \tl_if_empty:NF \g_nicematrix_code_before_tl
3911 {
3912   \tl_gput_right:Ne \g_@@_aux_tl
3913   {
3914     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3915     { \exp_not:o \g_nicematrix_code_before_tl }
3916   }
3917   \tl_gclear:N \g_nicematrix_code_before_tl
3918 }

3919 \str_gclear:N \g_@@_name_env_str
3920 \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the array¹³. 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`.

```
3921 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3922 }

3923 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
3924 {
3925   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3926   \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }
```

¹³e.g. `\color[rgb]{0.5,0.5,0}`

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

```

3927 \dim_set:Nn \l_@@_xdots_shorten_start_dim
3928 { 0.6 \l_@@_xdots_shorten_start_dim }
3929 \dim_set:Nn \l_@@_xdots_shorten_end_dim
3930 { 0.6 \l_@@_xdots_shorten_end_dim }
3931 }

```

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`.

```

3932 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3933 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

3934 \cs_new_protected:Npn \@@_create_alias_nodes:
3935 {
3936   \int_step_inline:nn { \c@iRow }
3937   {
3938     \pgfnodealias
3939     { \l_@@_name_str - ##1 - last }
3940     { \@@_env: - ##1 - \int_use:N \c@jCol }
3941   }
3942   \int_step_inline:nn { \c@jCol }
3943   {
3944     \pgfnodealias
3945     { \l_@@_name_str - last - ##1 }
3946     { \@@_env: - \int_use:N \c@iRow - ##1 }
3947   }
3948   \pgfnodealias % added 2025-04-05
3949   { \l_@@_name_str - last - last }
3950   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
3951 }

```

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).

```

3952 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3953 {
3954   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3955   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3956 }

```

The following command must *not* be protected.

```

3957 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3958 {
3959   { #1 }
3960   { #2 }
3961   {
3962     \int_compare:nNnTF { #3 } > { 98 }
3963     { \int_use:N \c@iRow }
3964     { #3 }
3965   }
3966   {
3967     \int_compare:nNnTF { #4 } > { 98 }
3968     { \int_use:N \c@jCol }
3969     { #4 }
3970   }

```

```

3971 { #5 }
3972 }

```

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).

```

3973 \hook_gput_code:nnn { begindocument } { . }
3974 {
3975   \cs_new_protected:Npe \@@_draw_dotted_lines:
3976   {
3977     \c_@@_pgfortikzpicture_tl
3978     \@@_draw_dotted_lines_i:
3979     \c_@@_endpgfortikzpicture_tl
3980   }
3981 }

```

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

```

3982 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3983 {
3984   \pgfrememberpicturepositiononpagetrue
3985   \pgf@relevantforpicturesizefalse
3986   \g_@@_HVdotsfor_lines_tl
3987   \g_@@_Vdots_lines_tl
3988   \g_@@_Ddots_lines_tl
3989   \g_@@_Iddots_lines_tl
3990   \g_@@_Cdots_lines_tl
3991   \g_@@_Ldots_lines_tl
3992 }

3993 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3994 {
3995   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3996   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3997 }

```

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

```

3998 \pgfdeclareshape { @@_diag_node }
3999 {
4000   \savedanchor { \five }
4001   {
4002     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4003     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4004   }
4005   \anchor { 5 } { \five }
4006   \anchor { center } { \pgfpointorigin }
4007   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4008   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4009   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4010   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4011   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4012   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4013   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4014   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4015   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4016   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4017 }

```

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).

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.

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

16 We draw the dotted lines

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

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

The command `\@@_find_extremities_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.

```
4059 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4060 {
```

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

```
4061 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4062 \int_set:Nn \l_@@_initial_i_int { #1 }
4063 \int_set:Nn \l_@@_initial_j_int { #2 }
4064 \int_set:Nn \l_@@_final_i_int { #1 }
4065 \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.

```
4066 \bool_set_false:N \l_@@_stop_loop_bool
4067 \bool_do_until:Nn \l_@@_stop_loop_bool
4068 {
4069   \int_add:Nn \l_@@_final_i_int { #3 }
4070   \int_add:Nn \l_@@_final_j_int { #4 }
4071   \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.

```
4072   \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4073     \if_int_compare:w #3 = \c_one_int
4074       \bool_set_true:N \l_@@_final_open_bool
4075     \else:
4076       \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4077         \bool_set_true:N \l_@@_final_open_bool
4078       \fi:
4079     \fi:
4080   \else:
4081     \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4082       \if_int_compare:w #4 = -1
4083         \bool_set_true:N \l_@@_final_open_bool
4084       \fi:
4085     \else:
4086       \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4087         \if_int_compare:w #4 = \c_one_int
4088           \bool_set_true:N \l_@@_final_open_bool
4089         \fi:
4090       \fi:
4091     \fi:
4092   \fi:
4093   \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.

```
4094     {
```

We do a step backwards.

```
4095         \int_sub:Nn \l_@@_final_i_int { #3 }
4096         \int_sub:Nn \l_@@_final_j_int { #4 }
4097         \bool_set_true:N \l_@@_stop_loop_bool
4098     }
```

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`.

```
4099     {
4100         \cs_if_exist:cTF
4101         {
4102             @@ _ dotted _
4103             \int_use:N \l_@@_final_i_int -
4104             \int_use:N \l_@@_final_j_int
4105         }
4106         {
4107             \int_sub:Nn \l_@@_final_i_int { #3 }
4108             \int_sub:Nn \l_@@_final_j_int { #4 }
4109             \bool_set_true:N \l_@@_final_open_bool
4110             \bool_set_true:N \l_@@_stop_loop_bool
4111         }
4112         {
4113             \cs_if_exist:cTF
4114             {
4115                 pgf @ sh @ ns @ \@@_env:
4116                 - \int_use:N \l_@@_final_i_int
4117                 - \int_use:N \l_@@_final_j_int
4118             }
4119             { \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.

```
4120         {
4121             \cs_set_nopar:cpn
4122             {
4123                 @@ _ dotted _
4124                 \int_use:N \l_@@_final_i_int -
4125                 \int_use:N \l_@@_final_j_int
4126             }
4127             { }
4128         }
4129     }
4130 }
4131 }
```

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

```
4132     \bool_set_false:N \l_@@_stop_loop_bool
```

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

```
4133     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4134     \bool_do_until:Nn \l_@@_stop_loop_bool
4135     {
4136         \int_sub:Nn \l_@@_initial_i_int { #3 }
4137         \int_sub:Nn \l_@@_initial_j_int { #4 }
4138         \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.

```

4139         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4140         \if_int_compare:w #3 = \c_one_int
4141         \bool_set_true:N \l_@@_initial_open_bool
4142         \else:
\l_tmpa_int contains \l_@@_col_min_int - 1 (only for efficiency).
4143         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4144         \bool_set_true:N \l_@@_initial_open_bool
4145         \fi:
4146     \fi:
4147 \else:
4148     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4149     \if_int_compare:w #4 = \c_one_int
4150     \bool_set_true:N \l_@@_initial_open_bool
4151     \fi:
4152 \else:
4153     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4154     \if_int_compare:w #4 = -1
4155     \bool_set_true:N \l_@@_initial_open_bool
4156     \fi:
4157     \fi:
4158 \fi:
4159 \fi:
4160 \bool_if:NTF \l_@@_initial_open_bool
4161 {
4162     \int_add:Nn \l_@@_initial_i_int { #3 }
4163     \int_add:Nn \l_@@_initial_j_int { #4 }
4164     \bool_set_true:N \l_@@_stop_loop_bool
4165 }
4166 {
4167     \cs_if_exist:cTF
4168     {
4169         @@ _ dotted _
4170         \int_use:N \l_@@_initial_i_int -
4171         \int_use:N \l_@@_initial_j_int
4172     }
4173     {
4174         \int_add:Nn \l_@@_initial_i_int { #3 }
4175         \int_add:Nn \l_@@_initial_j_int { #4 }
4176         \bool_set_true:N \l_@@_initial_open_bool
4177         \bool_set_true:N \l_@@_stop_loop_bool
4178     }
4179     {
4180         \cs_if_exist:cTF
4181         {
4182             pgf @ sh @ ns @ \@@_env:
4183             - \int_use:N \l_@@_initial_i_int
4184             - \int_use:N \l_@@_initial_j_int
4185         }
4186         { \bool_set_true:N \l_@@_stop_loop_bool }
4187         {
4188             \cs_set_nopar:cpn
4189             {
4190                 @@ _ dotted _
4191                 \int_use:N \l_@@_initial_i_int -
4192                 \int_use:N \l_@@_initial_j_int
4193             }
4194             { }
4195         }
4196     }
4197 }

```



```
4198 }
```

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.

```
4199 \seq_gput_right:N\g_@@_pos_of_xdots_seq
4200 {
4201   { \int_use:N \l_@@_initial_i_int }
```

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

```
4202   { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4203   { \int_use:N \l_@@_final_i_int }
4204   { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4205   { } % for the name of the block
4206 }
4207 }
```

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

```
4208 \cs_new_protected:Npn \@@_open_shorten:
4209 {
4210   \bool_if:NT \l_@@_initial_open_bool
4211   { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4212   \bool_if:NT \l_@@_final_open_bool
4213   { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4214 }
```

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).

```
4215 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4216 {
4217   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4218   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4219   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4220   \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`.

```
4221 \seq_if_empty:NF \g_@@_submatrix_seq
4222 {
4223   \seq_map_inline:Nn \g_@@_submatrix_seq
4224   { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
4225 }
4226 }
```

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

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

```
\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
{
  \bool_if:nT
  {
    \int_compare_p:n { #3 <= #1 <= #5 }
    &&
    \int_compare_p:n { #4 <= #2 <= #6 }
```

```

}
{
  \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
  \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
  \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
  \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
}
}

```

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

```

4227 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4228 {
4229   \if_int_compare:w #3 > #1
4230   \else:
4231     \if_int_compare:w #1 > #5
4232     \else:
4233       \if_int_compare:w #4 > #2
4234       \else:
4235         \if_int_compare:w #2 > #6
4236         \else:
4237           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4238           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4239           \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4240           \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4241         \fi:
4242       \fi:
4243     \fi:
4244   \fi:
4245 }

4246 \cs_new_protected:Npn \@@_set_initial_coords:
4247 {
4248   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4249   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4250 }
4251 \cs_new_protected:Npn \@@_set_final_coords:
4252 {
4253   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4254   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4255 }
4256 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4257 {
4258   \pgfpointanchor
4259   {
4260     \@@_env:
4261     - \int_use:N \l_@@_initial_i_int
4262     - \int_use:N \l_@@_initial_j_int
4263   }
4264   { #1 }
4265   \@@_set_initial_coords:
4266 }
4267 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4268 {
4269   \pgfpointanchor
4270   {
4271     \@@_env:
4272     - \int_use:N \l_@@_final_i_int
4273     - \int_use:N \l_@@_final_j_int
4274   }
4275   { #1 }
4276   \@@_set_final_coords:
4277 }

```

```

4278 \cs_new_protected:Npn \@@_open_x_initial_dim:
4279 {
4280   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4281   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4282   {
4283     \cs_if_exist:cT
4284     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4285     {
4286       \pgfpointanchor
4287       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4288       { west }
4289       \dim_set:Nn \l_@@_x_initial_dim
4290       { \dim_min:nn { \l_@@_x_initial_dim } { \pgf@x } }
4291     }
4292   }

```

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

```

4293   \dim_compare:nNnT { \l_@@_x_initial_dim } = { \c_max_dim }
4294   {
4295     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4296     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4297     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4298   }
4299 }

4300 \cs_new_protected:Npn \@@_open_x_final_dim:
4301 {
4302   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4303   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4304   {
4305     \cs_if_exist:cT
4306     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4307     {
4308       \pgfpointanchor
4309       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4310       { east }
4311       \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
4312       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4313     }
4314   }

```

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

```

4315   \dim_compare:nNnT { \l_@@_x_final_dim } = { - \c_max_dim }
4316   {
4317     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4318     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4319     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4320   }
4321 }

```

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.

```

4322 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4323 {
4324   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4325   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4326   {
4327     \@@_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.

```

4328   \group_begin:
4329   \@@_open_shorten:

```

```

4330     \int_if_zero:nTF { #1 }
4331     { \color { nicematrix-first-row } }
4332     {

```

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.

```

4333         \int_compare:nNtT { #1 } = { \l_@@_last_row_int }
4334         { \color { nicematrix-last-row } }
4335     }
4336     \keys_set:nn { nicematrix / xdots } { #3 }
4337     \@@_color:o \l_@@_xdots_color_tl
4338     \@@_actually_draw_Ldots:
4339     \group_end:
4340 }
4341 }

```

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`.

```

4342 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4343 {
4344     \bool_if:NTF \l_@@_initial_open_bool
4345     {
4346         \@@_open_x_initial_dim:
4347         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4348         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4349     }
4350     { \@@_set_initial_coords_from_anchor:n { base-east } }
4351     \bool_if:NTF \l_@@_final_open_bool
4352     {
4353         \@@_open_x_final_dim:
4354         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4355         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4356     }
4357     { \@@_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.

```

4358     \bool_lazy_all:nTF
4359     {
4360         \l_@@_initial_open_bool
4361         \l_@@_final_open_bool
4362         { \int_compare_p:nNn { \l_@@_initial_i_int } = { \l_@@_last_row_int } }
4363     }
4364     {
4365         \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4366         \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4367     }

```

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

```

4368     {
4369         \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim

```

```

4370     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4371   }
4372   \@@_draw_line:
4373 }

```

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.

```

4374 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4375 {
4376   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4377   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4378   {
4379     \@@_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.

```

4380     \group_begin:
4381     \@@_open_shorten:
4382     \int_if_zero:nTF { #1 }
4383     { \color { nicematrix-first-row } }
4384     {

```

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.

```

4385         \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4386         { \color { nicematrix-last-row } }
4387     }
4388     \keys_set:nn { nicematrix / xdots } { #3 }
4389     \@@_color:o \l_@@_xdots_color_tl
4390     \@@_actually_draw_Cdots:
4391   \group_end:
4392 }
4393 }

```

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`.

```

4394 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4395 {
4396   \bool_if:NTF \l_@@_initial_open_bool
4397   { \@@_open_x_initial_dim: }
4398   { \@@_set_initial_coords_from_anchor:n { mid-east } }
4399   \bool_if:NTF \l_@@_final_open_bool
4400   { \@@_open_x_final_dim: }
4401   { \@@_set_final_coords_from_anchor:n { mid-west } }
4402   \bool_lazy_and:nnTF
4403   { \l_@@_initial_open_bool }
4404   { \l_@@_final_open_bool }
4405   {
4406     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4407     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4408     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4409     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }

```

```

4410     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4411   }
4412   {
4413     \bool_if:NT \l_@@_initial_open_bool
4414     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4415     \bool_if:NT \l_@@_final_open_bool
4416     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4417   }
4418   \@@_draw_line:
4419 }

4420 \cs_new_protected:Npn \@@_open_y_initial_dim:
4421 {
4422   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4423   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4424   {
4425     \cs_if_exist:cT
4426     { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4427     {
4428       \pgfpointanchor
4429       { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4430       { north }
4431       \dim_compare:nNnT { \pgf@y } > { \l_@@_y_initial_dim }
4432       { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4433     }
4434   }
4435   \dim_compare:nNnT { \l_@@_y_initial_dim } = { - \c_max_dim }
4436   {
4437     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4438     \dim_set:Nn \l_@@_y_initial_dim
4439     {
4440       \fp_to_dim:n
4441       {
4442         \pgf@y
4443         + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4444       }
4445     }
4446   }
4447 }

4448 \cs_new_protected:Npn \@@_open_y_final_dim:
4449 {
4450   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4451   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4452   {
4453     \cs_if_exist:cT
4454     { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4455     {
4456       \pgfpointanchor
4457       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4458       { south }
4459       \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
4460       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4461     }
4462   }
4463   \dim_compare:nNnT { \l_@@_y_final_dim } = { \c_max_dim }
4464   {
4465     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4466     \dim_set:Nn \l_@@_y_final_dim
4467     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4468   }
4469 }

```

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.

```

4470 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4471 {
4472   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4473   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4474   {
4475     \@@_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.

```

4476     \group_begin:
4477     \@@_open_shorten:
4478     \int_if_zero:nTF { #2 }
4479     { \color { nicematrix-first-col } }
4480     {
4481       \int_compare:nNnT { #2 } = { \l_@@_last_col_int }
4482       { \color { nicematrix-last-col } }
4483     }
4484     \keys_set:nn { nicematrix / xdots } { #3 }
4485     \@@_color:o \l_@@_xdots_color_tl
4486     \bool_if:NTF \l_@@_Vbrace_bool
4487     { \@@_actually_draw_Vbrace: }
4488     { \@@_actually_draw_Vdots: }
4489     \group_end:
4490   }
4491 }

```

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`.

```

4492 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4493 {
4494   \bool_lazy_and:nnTF { \l_@@_initial_open_bool } { \l_@@_final_open_bool }
4495   { \@@_actually_draw_Vdots_i: }
4496   { \@@_actually_draw_Vdots_ii: }
4497   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4498   \@@_draw_line:
4499 }

```

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

```

4500 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4501 {
4502   \@@_open_y_initial_dim:
4503   \@@_open_y_final_dim:
4504   \int_if_zero:nTF { \l_@@_initial_j_int }

```

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

```

4505   {
4506     \@@_qpoint:n { col - 1 }
4507     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4508     \dim_sub:Nn \l_@@_x_initial_dim
4509     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4510   }
4511   {

```

```

4512 \bool_lazy_and:nnTF
4513 { \int_compare_p:nNn { \l_@@_last_col_int } > { -2 } }
4514 {
4515   \int_compare_p:nNn
4516   { \l_@@_initial_j_int } = { \g_@@_col_total_int }
4517 }

```

We have a dotted line open on both sides and which is in the “last column”.

```

4518 {
4519   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4520   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4521   \dim_add:Nn \l_@@_x_initial_dim
4522   { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4523 }

```

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

```

4524 {
4525   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4526   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4527   \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4528   \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4529 }
4530 }
4531 }

```

The command `\@@_draw_line:` is in `\@@_actually_draw_Vdots:`

Now, the dotted line is *not* open on both sides (maybe open on only one side).

The main task is to determine the x -value of the dotted line to draw.

The boolean `\l_tmpa_bool` will indicate whether the column is of type `l` or may be considered as if.

```

4532 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4533 {
4534   \bool_set_false:N \l_tmpa_bool
4535   \bool_if:NF \l_@@_initial_open_bool
4536   {
4537     \bool_if:NF \l_@@_final_open_bool
4538     {
4539       \@@_set_initial_coords_from_anchor:n { south~west }
4540       \@@_set_final_coords_from_anchor:n { north~west }
4541       \bool_set:Nn \l_tmpa_bool
4542       {
4543         \dim_compare_p:nNn
4544         { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4545       }
4546     }
4547   }

```

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

```

4548 \bool_if:NTF \l_@@_initial_open_bool
4549 {
4550   \@@_open_y_initial_dim:
4551   \@@_set_final_coords_from_anchor:n { north }
4552   \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4553 }
4554 {
4555   \@@_set_initial_coords_from_anchor:n { south }
4556   \bool_if:NTF \l_@@_final_open_bool
4557   { \@@_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.

```

4558 {
4559   \@@_set_final_coords_from_anchor:n { north }
4560   \dim_compare:nNnF { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4561   {

```



```

4562         \dim_set:Nn \l_@@_x_initial_dim
4563         {
4564             \bool_if:NTF \l_tmpa_bool { \dim_min:nn } { \dim_max:nn }
4565             \l_@@_x_initial_dim \l_@@_x_final_dim
4566         }
4567     }
4568 }
4569 }
4570 }

```

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`.

```

4571 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4572 {
4573     \bool_if:NTF \l_@@_initial_open_bool
4574     { \@@_open_y_initial_dim: }
4575     { \@@_set_initial_coords_from_anchor:n { south } }
4576     \bool_if:NTF \l_@@_final_open_bool
4577     { \@@_open_y_final_dim: }
4578     { \@@_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.

```

4579     \int_if_zero:nTF { \l_@@_initial_j_int }
4580     {
4581         \@@_qpoint:n { col - 1 }
4582         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4583         \dim_sub:Nn \l_@@_x_initial_dim
4584         { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4585     }

```

Elsewhere, the brace must be drawn left flush.

```

4586     {
4587         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4588         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4589         \dim_add:Nn \l_@@_x_initial_dim
4590         { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4591     }

```

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

```

4592     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4593     \@@_draw_line:
4594 }

```

```

4595 \cs_new:Npn \@@_colsep:
4596 { \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.

```

4597 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4598 {
4599   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4600   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4601   {
4602     \@@_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.

```

4603   \group_begin:
4604     \@@_open_shorten:
4605     \keys_set:nn { nicematrix / xdots } { #3 }
4606     \@@_color:o \l_@@_xdots_color_tl
4607     \@@_actually_draw_Ddots:
4608   \group_end:
4609 }
4610 }

```

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.`

```

4611 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4612 {
4613   \bool_if:NTF \l_@@_initial_open_bool
4614   {
4615     \@@_open_y_initial_dim:
4616     \@@_open_x_initial_dim:
4617   }
4618   { \@@_set_initial_coords_from_anchor:n { south-east } }
4619   \bool_if:NTF \l_@@_final_open_bool
4620   {
4621     \@@_open_x_final_dim:
4622     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4623   }
4624   { \@@_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.

```

4625   \bool_if:NT \l_@@_parallelize_diags_bool
4626   {
4627     \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).

```

4628     \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 Δ_x and the Δ_y of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```

4629     {

```

```

4630      \dim_gset:Nn \g_@@_delta_x_one_dim
4631      { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4632      \dim_gset:Nn \g_@@_delta_y_one_dim
4633      { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4634    }

```

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`.

```

4635    {
4636      \dim_compare:nNnF { \g_@@_delta_x_one_dim } = { \c_zero_dim }
4637      {
4638        \dim_set:Nn \l_@@_y_final_dim
4639        {
4640          \l_@@_y_initial_dim +
4641          ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4642          \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4643        }
4644      }
4645    }
4646  }
4647  \@@_draw_line:
4648 }

```

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.

```

4649 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4650 {
4651   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4652   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4653   {
4654     \@@_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.

```

4655     \group_begin:
4656     \@@_open_shorten:
4657     \keys_set:nn { nicematrix / xdots } { #3 }
4658     \@@_color:o \l_@@_xdots_color_tl
4659     \@@_actually_draw_Iddots:
4660   \group_end:
4661 }
4662 }

```

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`.

```

4663 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4664 {
4665   \bool_if:NTF \l_@@_initial_open_bool
4666   {
4667     \@@_open_y_initial_dim:
4668     \@@_open_x_initial_dim:
4669   }

```

```

4670     { \@@_set_initial_coords_from_anchor:n { south-west } }
4671 \bool_if:NTF \l_@@_final_open_bool
4672 {
4673     \@@_open_y_final_dim:
4674     \@@_open_x_final_dim:
4675 }
4676 { \@@_set_final_coords_from_anchor:n { north-east } }
4677 \bool_if:NT \l_@@_parallelize_diags_bool
4678 {
4679     \int_gincr:N \g_@@_iddots_int
4680     \int_compare:nNnTF { \g_@@_iddots_int } = { \c_one_int }
4681     {
4682         \dim_gset:Nn \g_@@_delta_x_two_dim
4683         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4684         \dim_gset:Nn \g_@@_delta_y_two_dim
4685         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4686     }
4687     {
4688         \dim_compare:nNnF { \g_@@_delta_x_two_dim } = { \c_zero_dim }
4689         {
4690             \dim_set:Nn \l_@@_y_final_dim
4691             {
4692                 \l_@@_y_initial_dim +
4693                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4694                 \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4695             }
4696         }
4697     }
4698 }
4699 \@@_draw_line:
4700 }

```

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`

```

4701 \cs_new_protected:Npn \@@_draw_line:
4702 {
4703     \pgfrememberpicturepositiononpagetrue
4704     \pgf@relevantforpicturesizefalse
4705     \bool_lazy_or:nnTF
4706     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4707     { \l_@@_dotted_bool }
4708     { \@@_draw_standard_dotted_line: }
4709     { \@@_draw_unstandard_dotted_line: }
4710 }

```

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.

```

4711 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4712 {
4713   \begin { scope }
4714   \@@_draw_unstandard_dotted_line:o
4715   { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4716 }

```

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.

```

4717 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4718 {
4719   \@@_draw_unstandard_dotted_line:nooo
4720   { #1 }
4721   \l_@@_xdots_up_tl
4722   \l_@@_xdots_down_tl
4723   \l_@@_xdots_middle_tl
4724 }
4725 \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.

```

4726 \hook_gput_code:nnn { begindocument } { . }
4727 {
4728   \IfPackageLoadedT { tikz }
4729   {
4730     \tikzset
4731     {
4732       @@_node_above / .style = { sloped , above } ,
4733       @@_node_below / .style = { sloped , below } ,
4734       @@_node_middle / .style =
4735       {
4736         sloped ,
4737         inner~sep = \c_@@_innersep_middle_dim
4738       }
4739     }
4740   }
4741 }

4742 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4743 {

```

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 ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

4744 \dim_zero_new:N \l_@@_l_dim
4745 \dim_set:Nn \l_@@_l_dim
4746 {
4747   \fp_to_dim:n
4748   {
4749     sqrt
4750     (
4751       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4752       +
4753       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4754     )
4755   }
4756 }

```

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.

```

4757 \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4758 {
4759   \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4760   \@@_draw_unstandard_dotted_line_i:
4761 }

```

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

```

4762 \bool_if:NT \l_@@_xdots_h_labels_bool
4763 {
4764   \tikzset
4765   {
4766     @@_node_above / .style = { auto = left } ,
4767     @@_node_below / .style = { auto = right } ,
4768     @@_node_middle / .style = { innersep = \c_@@_innersep_middle_dim }
4769   }
4770 }
4771 \tl_if_empty:nF { #4 }
4772 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4773 \draw
4774 [ #1 ]
4775 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )

```

Be careful: We can't put `\c_math_toggle_token` instead of `$` in the following lines because we are in the contents of Tikz nodes (and they will be *rescanned* if the Tikz library `babel` is loaded).

```

4776 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4777 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4778 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4779 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4780 \end { scope }
4781 }
4782 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4783 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4784 {
4785   \dim_set:Nn \l_tmpa_dim
4786   {
4787     \l_@@_x_initial_dim
4788     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4789     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4790   }
4791   \dim_set:Nn \l_tmpb_dim
4792   {
4793     \l_@@_y_initial_dim
4794     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4795     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4796   }
4797   \dim_set:Nn \l_@@_tmpc_dim
4798   {
4799     \l_@@_x_final_dim
4800     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4801     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4802   }
4803   \dim_set:Nn \l_@@_tmpd_dim
4804   {
4805     \l_@@_y_final_dim
4806     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4807     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4808   }
4809   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4810   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim

```

```

4811 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4812 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4813 }

```

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

```

4814 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4815 {
4816   \group_begin:

```

The dimension `\l_@@_l_dim` is the length ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

4817   \dim_zero_new:N \l_@@_l_dim
4818   \dim_set:Nn \l_@@_l_dim
4819   {
4820     \fp_to_dim:n
4821     {
4822       sqrt
4823       (
4824         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4825         +
4826         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4827       )
4828     }
4829   }

```

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.

```

4830   \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4831   {
4832     \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4833     { \@@_draw_standard_dotted_line_i: }
4834   }
4835   \group_end:
4836   \bool_lazy_all:nF
4837   {
4838     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4839     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4840     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4841   }
4842   { \@@_labels_standard_dotted_line: }
4843 }
4844 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4845 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4846 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

4847   \int_set:Nn \l_tmpa_int
4848   {
4849     \dim_ratio:nn
4850     {
4851       \l_@@_l_dim
4852       - \l_@@_xdots_shorten_start_dim
4853       - \l_@@_xdots_shorten_end_dim
4854     }
4855     { \l_@@_xdots_inter_dim }
4856   }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

4857 \dim_set:Nn \l_tmpa_dim
4858 {
4859   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4860   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4861 }
4862 \dim_set:Nn \l_tmpb_dim
4863 {
4864   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4865   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4866 }

```

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.

```

4867 \dim_gadd:Nn \l_@@_x_initial_dim
4868 {
4869   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4870   \dim_ratio:nn
4871   {
4872     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4873     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4874   }
4875   { 2 \l_@@_l_dim }
4876 }
4877 \dim_gadd:Nn \l_@@_y_initial_dim
4878 {
4879   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4880   \dim_ratio:nn
4881   {
4882     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4883     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4884   }
4885   { 2 \l_@@_l_dim }
4886 }
4887 \pgf@relevantforpicturesizefalse
4888 \int_step_inline:nnn { \c_zero_int } { \l_tmpa_int }
4889 {
4890   \pgfpathcircle
4891   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4892   { \l_@@_xdots_radius_dim }
4893   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4894   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4895 }
4896 \pgfusepathqfill
4897 }

```

```

4898 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
4899 {
4900   \pgfscope
4901   \pgftransformshift
4902   {
4903     \pgfpointlineattime { 0.5 }
4904     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4905     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4906   }
4907   \fp_set:Nn \l_tmpa_fp
4908   {
4909     atand
4910     (
4911       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4912       \l_@@_x_final_dim - \l_@@_x_initial_dim
4913     )

```



```

4914     }
4915     \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4916     \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4917     \tl_if_empty:NF \l_@@_xdots_middle_tl
4918     {
4919         \begin { pgfscope }
4920         \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4921         \pgfnode
4922         { rectangle }
4923         { center }
4924         {
4925             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4926             {
4927                 \c_math_toggle_token
4928                 \scriptstyle \l_@@_xdots_middle_tl
4929                 \c_math_toggle_token
4930             }
4931         }
4932         { }
4933         {
4934             \pgfsetfillcolor { white }
4935             \pgfusepath { fill }
4936         }
4937         \end { pgfscope }
4938     }
4939     \tl_if_empty:NF \l_@@_xdots_up_tl
4940     {
4941         \pgfnode
4942         { rectangle }
4943         { south }
4944         {
4945             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4946             {
4947                 \c_math_toggle_token
4948                 \scriptstyle \l_@@_xdots_up_tl
4949                 \c_math_toggle_token
4950             }
4951         }
4952         { }
4953         { \pgfusepath { } }
4954     }
4955     \tl_if_empty:NF \l_@@_xdots_down_tl
4956     {
4957         \pgfnode
4958         { rectangle }
4959         { north }
4960         {
4961             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4962             {
4963                 \c_math_toggle_token
4964                 \scriptstyle \l_@@_xdots_down_tl
4965                 \c_math_toggle_token
4966             }
4967         }
4968         { }
4969         { \pgfusepath { } }
4970     }
4971     \endpgfscope
4972 }

```

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.

```
4973 \hook_gput_code:nnn { begindocument } { . }
4974 {
```

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

```
4975 \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
4976 \cs_new_protected:Npn \@@_Ldots:
4977 { \@@_collect_options:n { \@@_Ldots_i } }
4978 \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4979 {
4980   \int_if_zero:nTF { \c@jCol }
4981   { \@@_error:nn { in~first~col } { \Ldots } }
4982   {
4983     \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
4984     { \@@_error:nn { in~last~col } { \Ldots } }
4985     {
4986       \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
4987       { #1 , down = #2 , up = #3 , middle = #4 }
4988     }
4989   }
4990   \bool_if:NF \l_@@_nullify_dots_bool
4991   { \phantom { \ensuremath { \@@_old_ldots: } } }
4992   \bool_gset_true:N \g_@@_empty_cell_bool
4993 }
```

```
4994 \cs_new_protected:Npn \@@_Cdots:
4995 { \@@_collect_options:n { \@@_Cdots_i } }
4996 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4997 {
4998   \int_if_zero:nTF { \c@jCol }
4999   { \@@_error:nn { in~first~col } { \Cdots } }
5000   {
5001     \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5002     { \@@_error:nn { in~last~col } { \Cdots } }
5003     {
5004       \@@_instruction_of_type:nnn { \c_false_bool } { \Cdots }
5005       { #1 , down = #2 , up = #3 , middle = #4 }
5006     }
5007   }
5008   \bool_if:NF \l_@@_nullify_dots_bool
5009   { \phantom { \ensuremath { \@@_old_cdots: } } }
5010   \bool_gset_true:N \g_@@_empty_cell_bool
5011 }
```

```
5012 \cs_new_protected:Npn \@@_Vdots:
5013 { \@@_collect_options:n { \@@_Vdots_i } }
5014 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5015 {
5016   \int_if_zero:nTF { \c@iRow }
```

```

5017     { \@@_error:nn { in~first~row } { \Vdots } }
5018   {
5019     \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
5020     { \@@_error:nn { in~last~row } { \Vdots } }
5021     {
5022       \@@_instruction_of_type:nnn { \c_false_bool } { \Vdots }
5023       { #1 , down = #2 , up = #3 , middle = #4 }
5024     }
5025   }
5026   \bool_if:NF \l_@@_nullify_dots_bool
5027   { \phantom { \ensuremath { \@@_old_vdots: } } }
5028   \bool_gset_true:N \g_@@_empty_cell_bool
5029 }

5030 \cs_new_protected:Npn \@@_Ddots:
5031 { \@@_collect_options:n { \@@_Ddots_i } }
5032 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5033 {
5034   \int_case:nnF \c@iRow
5035   {
5036     0 { \@@_error:nn { in~first~row } { \Ddots } }
5037     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5038   }
5039   {
5040     \int_case:nnF \c@jCol
5041     {
5042       0 { \@@_error:nn { in~first~col } { \Ddots } }
5043       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5044     }
5045     {
5046       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5047       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
5048       { #1 , down = #2 , up = #3 , middle = #4 }
5049     }
5050   }
5051 }
5052 \bool_if:NF \l_@@_nullify_dots_bool
5053 { \phantom { \ensuremath { \@@_old_ddots: } } }
5054 \bool_gset_true:N \g_@@_empty_cell_bool
5055 }

5056 \cs_new_protected:Npn \@@_Iddots:
5057 { \@@_collect_options:n { \@@_Iddots_i } }
5058 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5059 {
5060   \int_case:nnF \c@iRow
5061   {
5062     0 { \@@_error:nn { in~first~row } { \Iddots } }
5063     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5064   }
5065   {
5066     \int_case:nnF \c@jCol
5067     {
5068       0 { \@@_error:nn { in~first~col } { \Iddots } }
5069       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5070     }
5071     {
5072       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5073       \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { \Iddots }
5074       { #1 , down = #2 , up = #3 , middle = #4 }
5075     }
5076   }

```

```

5077     \bool_if:NF \l_@@_nullify_dots_bool
5078     { \phantom { \ensuremath { \@@_old_iddots: } } }
5079     \bool_gset_true:N \g_@@_empty_cell_bool
5080   }
5081 }

```

End of the \AddToHook.

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

```

5082 \keys_define:nn { nicematrix / Ddots }
5083 {
5084   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5085   draw-first .default:n = true ,
5086   draw-first .value_forbidden:n = true
5087 }

```

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

```

5088 \cs_new_protected:Npn \@@_Hspace:
5089 {
5090   \bool_gset_true:N \g_@@_empty_cell_bool
5091   \hspace
5092 }

```

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`.

```

5093 \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.

```

5094 \cs_new:Npn \@@_Hdotsfor:
5095 {
5096   \bool_lazy_and:nnTF
5097   { \int_if_zero_p:n { \c@jCol } }
5098   { \int_if_zero_p:n { \l_@@_first_col_int } }
5099   {
5100     \bool_if:NTF \g_@@_after_col_zero_bool
5101     {
5102       \multicolumn { 1 } { c } { }
5103       \@@_Hdotsfor_i:
5104     }
5105     { \@@_fatal:n { Hdotsfor~in~col~0 } }
5106   }
5107   {
5108     \multicolumn { 1 } { c } { }
5109     \@@_Hdotsfor_i:
5110   }
5111 }

```

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:).

```

5112 \hook_gput_code:nnn { begindocument } { . }
5113 {

```

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

```

5114   \cs_new_protected:Npn \@@_Hdotsfor_i:
5115   { \@@_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`).

```

5116 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5117 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5118 {
5119   \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5120   {
5121     \@@_Hdotsfor:nnnn
5122     { \int_use:N \c@iRow }
5123     { \int_use:N \c@jCol }
5124     { #2 }
5125     {
5126       #1 , #3 ,
5127       down = \exp_not:n { #4 } ,
5128       up = \exp_not:n { #5 } ,
5129       middle = \exp_not:n { #6 }
5130     }
5131   }
5132   \prg_replicate:nn { #2 - 1 }
5133   {
5134     &
5135     \multicolumn { 1 } { c } { }
5136     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5137   }
5138 }
5139 }

```

```

5140 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5141 {
5142   \bool_set_false:N \l_@@_initial_open_bool
5143   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5144 \int_set:Nn \l_@@_initial_i_int { #1 }
5145 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5146 \int_compare:nNnTF { #2 } = { \c_one_int }
5147 {
5148   \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5149   \bool_set_true:N \l_@@_initial_open_bool
5150 }
5151 {
5152   \cs_if_exist:cTF
5153   {
5154     pgf @ sh @ ns @ \@@_env:
5155     - \int_use:N \l_@@_initial_i_int
5156     - \int_eval:n { #2 - 1 }
5157   }
5158   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5159   {
5160     \int_set:Nn \l_@@_initial_j_int { #2 }
5161     \bool_set_true:N \l_@@_initial_open_bool
5162   }
5163 }
5164 \int_compare:nNnTF { #2 + #3 - 1 } = { \c@jCol }
5165 {
5166   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5167   \bool_set_true:N \l_@@_final_open_bool
5168 }
5169 {
5170   \cs_if_exist:cTF
5171   {
5172     pgf @ sh @ ns @ \@@_env:

```

```

5173         - \int_use:N \l_@@_final_i_int
5174         - \int_eval:n { #2 + #3 }
5175     }
5176     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5177     {
5178         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5179         \bool_set_true:N \l_@@_final_open_bool
5180     }
5181 }
5182 \group_begin:
5183 \@@_open_shorten:
5184 \int_if_zero:nTF { #1 }
5185   { \color { nicematrix-first-row } }
5186   {
5187     \int_compare:nNnT { #1 } = { \g_@@_row_total_int }
5188     { \color { nicematrix-last-row } }
5189   }
5190 \keys_set:nn { nicematrix / xdots } { #4 }
5191 \@@_color:o \l_@@_xdots_color_tl
5192 \@@_actually_draw_Ldots:
5193 \group_end:

```

We declare all the cells concerned by the `\Hdotsfor` as “dotted” (for the dotted lines created by `\Cdots`, `\Ldots`, etc., this job is done by `\@@_find_extremities_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5194 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5195   { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5196 }
5197 \hook_gput_code:nnn { begindocument } { . }
5198 {
5199   \cs_new_protected:Npn \@@_Vdotsfor:
5200     { \@@_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`).

```

5201 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5202 \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5203 {
5204   \bool_gset_true:N \g_@@_empty_cell_bool
5205   \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5206   {
5207     \@@_Vdotsfor:nnnn
5208     { \int_use:N \c@iRow }
5209     { \int_use:N \c@jCol }
5210     { #2 }
5211     {
5212       #1 , #3 ,
5213       down = \exp_not:n { #4 } ,
5214       up = \exp_not:n { #5 } ,
5215       middle = \exp_not:n { #6 }
5216     }
5217   }
5218 }
5219 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5220 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5221 {
5222   \bool_set_false:N \l_@@_initial_open_bool
5223   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```
5224 \int_set:Nn \l_@@_initial_j_int { #2 }
5225 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int
```

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

```
5226 \int_compare:nNnTF { #1 } = { \c_one_int }
5227 {
5228   \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5229   \bool_set_true:N \l_@@_initial_open_bool
5230 }
5231 {
5232   \cs_if_exist:cTF
5233   {
5234     pgf @ sh @ ns @ \@@_env:
5235     - \int_eval:n { #1 - 1 }
5236     - \int_use:N \l_@@_initial_j_int
5237   }
5238   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5239   {
5240     \int_set:Nn \l_@@_initial_i_int { #1 }
5241     \bool_set_true:N \l_@@_initial_open_bool
5242   }
5243 }
5244 \int_compare:nNnTF { #1 + #3 - 1 } = { \c@iRow }
5245 {
5246   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5247   \bool_set_true:N \l_@@_final_open_bool
5248 }
5249 {
5250   \cs_if_exist:cTF
5251   {
5252     pgf @ sh @ ns @ \@@_env:
5253     - \int_eval:n { #1 + #3 }
5254     - \int_use:N \l_@@_final_j_int
5255   }
5256   { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5257   {
5258     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5259     \bool_set_true:N \l_@@_final_open_bool
5260   }
5261 }
5262 \group_begin:
5263 \@@_open_shorten:
5264 \int_if_zero:nTF { #2 }
5265 { \color { nicematrix-first-col } }
5266 {
5267   \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
5268   { \color { nicematrix-last-col } }
5269 }
5270 \keys_set:nn { nicematrix / xdots } { #4 }
5271 \@@_color:o \l_@@_xdots_color_tl
5272 \bool_if:NTF \l_@@_Vbrace_bool
5273 { \@@_actually_draw_Vbrace: }
5274 { \@@_actually_draw_Vdots: }
5275 \group_end:

5276 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5277 { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5278 }
```

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).

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

```

5279 \NewDocumentCommand \@@_rotate: { 0 { } }
5280 {
5281   \bool_gset_true:N \g_@@_rotate_bool
5282   \keys_set:nn { nicematrix / rotate } { #1 }
5283   \ignorespaces
5284 }

5285 \keys_define:nn { nicematrix / rotate }
5286 {
5287   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5288   c .value_forbidden:n = true ,
5289   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5290 }

```

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).¹⁴

```

5291 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5292 {
5293   \tl_if_empty:nTF { #2 }
5294     { #1 }
5295     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5296 }
5297 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5298 { \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).

```

5299 \hook_gput_code:nnn { begindocument } { . }
5300 {

```

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

```

5301   \tl_set_rescan:Nnn \l_tmpa_tl { }
5302   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5303   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5304   {
5305     \group_begin:
5306     \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5307     \@@_color:o \l_@@_xdots_color_tl
5308     \use:e
5309     {

```

¹⁴Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.


```

5310         \@@_line_i:nn
5311         { \@@_double_int_eval:n #2 - \q_stop }
5312         { \@@_double_int_eval:n #3 - \q_stop }
5313     }
5314     \group_end:
5315 }
5316 }

5317 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5318 {
5319     \bool_set_false:N \l_@@_initial_open_bool
5320     \bool_set_false:N \l_@@_final_open_bool
5321     \bool_lazy_or:nnTF
5322     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5323     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5324     { \@@_error:nnn { unknown-cell-for-line-in~CodeAfter } { #1 } { #2 } }

```

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

```

5325     { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5326 }

5327 \hook_gput_code:nnn { begindocument } { . }
5328 {
5329     \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5330     {

```

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:`.

```

5331     \c_@@_pgfortikzpicture_tl
5332     \@@_draw_line_iii:nn { #1 } { #2 }
5333     \c_@@_endpgfortikzpicture_tl
5334 }
5335 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5336 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5337 {
5338     \pgfrememberpicturepositiononpagetrue
5339     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5340     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5341     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5342     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5343     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5344     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5345     \@@_draw_line:
5346 }

```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don’t use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

20 The command `\RowStyle`

`\g_@@_row_style_tl` may contain several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

Then, `\g_@@_row_style_tl` will be inserted in all the cells of the array (and also in both components of a `\diagbox` in a cell of in a mono-row block).

The test `\@@_if_row_less_than:nn` ensures that the instructions are inserted only if you are in a row which is (still) in the scope of that instructions (which depends on the value of the key `nb-rows` of `\RowStyle`).

That test will be active even in an expandable context because `\@@_if_row_less_than:nn` is *not* protected.

`#1` is the first row *after* the scope of the instructions in `#2`

However, both arguments are implicit because they are taken by curryfication.

```
5347 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNnT { \c@iRow } < }
5348 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNnF { \c@jCol } < }
```

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

```
5349 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5350 {
5351   \tl_gput_right:Ne \g_@@_row_style_tl
5352   {
```

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

```
5353   \exp_not:N
5354   \@@_if_row_less_than:nn
5355   { \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`).

```
5356   {
5357     \exp_not:N
5358     \@@_if_col_greater_than:nn
5359     { \int_eval:n { \c@jCol } }
5360     { \exp_not:n { #1 } \scan_stop: }
5361   }
5362 }
5363 }
5364 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
```

```
5365 \keys_define:nn { nicematrix / RowStyle }
5366 {
5367   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5368   cell-space-top-limit .value_required:n = true ,
5369   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5370   cell-space-bottom-limit .value_required:n = true ,
5371   cell-space-limits .meta:n =
5372   {
5373     cell-space-top-limit = #1 ,
5374     cell-space-bottom-limit = #1 ,
5375   } ,
5376   color .tl_set:N = \l_@@_color_tl ,
5377   color .value_required:n = true ,
5378   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5379   bold .default:n = true ,
5380   nb-rows .code:n =
5381   \str_if_eq:eeTF { #1 } { * }
5382   { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5383   { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5384   nb-rows .value_required:n = true ,
5385   fill .tl_set:N = \l_@@_fill_tl ,
5386   fill .value_required:n = true ,
5387   opacity .tl_set:N = \l_@@_opacity_tl ,
5388   opacity .value_required:n = true ,
5389   rowcolor .tl_set:N = \l_@@_fill_tl ,
5390   rowcolor .value_required:n = true ,
5391   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5392   rounded-corners .default:n = 4 pt ,
5393   unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5394 }
```

```

5395 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5396 {
5397   \group_begin:
5398   \tl_clear:N \l_@@_fill_tl
5399   \tl_clear:N \l_@@_opacity_tl
5400   \tl_clear:N \l_@@_color_tl
5401   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5402   \dim_zero:N \l_@@_rounded_corners_dim
5403   \dim_zero:N \l_tmpa_dim
5404   \dim_zero:N \l_tmpb_dim
5405   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5406   \tl_if_empty:NF \l_@@_fill_tl
5407   {
5408     \@@_add_opacity_to_fill:
5409     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5410     {

```

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

```

5411       \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5412       { \int_use:N \c@iRow - \int_use:N \c@jCol }
5413       {
5414         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5415         - *
5416       }
5417       { \dim_use:N \l_@@_rounded_corners_dim }
5418     }
5419   }
5420   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5421   \dim_compare:nNnT { \l_tmpa_dim } > { \c_zero_dim }
5422   {
5423     \@@_put_in_row_style:e
5424     {
5425       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5426       {

```

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

```

5427         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5428         { \dim_use:N \l_tmpa_dim }
5429       }
5430     }
5431   }

```

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

```

5432   \dim_compare:nNnT { \l_tmpb_dim } > { \c_zero_dim }
5433   {
5434     \@@_put_in_row_style:e
5435     {
5436       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5437       {
5438         \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5439         { \dim_use:N \l_tmpb_dim }
5440       }
5441     }
5442   }

```

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

```

5443   \tl_if_empty:NF \l_@@_color_tl
5444   {
5445     \@@_put_in_row_style:e
5446     {
5447       \mode_leave_vertical:
5448       \@@_color:n { \l_@@_color_tl }

```

```

5449     }
5450 }
\l_@@_bold_row_style_bool is the value of the key bold.
5451 \bool_if:NT \l_@@_bold_row_style_bool
5452 {
5453   \@@_put_in_row_style:n
5454   {
5455     \exp_not:n
5456     {
5457       \if_mode_math:
5458         \c_math_toggle_token
5459         \bfseries \boldmath
5460         \c_math_toggle_token
5461       \else:
5462         \bfseries \boldmath
5463       \fi:
5464     }
5465   }
5466 }
5467 \group_end:
5468 \g_@@_row_style_tl
5469 \ignorespaces
5470 }

```

The following commande must *not* be protected.

```

5471 \cs_new:Npn \@@_rounded_from_row:n #1
5472 {
5473   \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

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

```

5474   { \int_eval:n { #1 } - 1 }
5475   {
5476     \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5477     - \exp_not:n { \int_use:N \c@jCol }
5478   }
5479   { \dim_use:N \l_@@_rounded_corners_dim }
5480 }

```

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).

```
5481 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5482 {
```

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.

```
5483 \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`.

```
5484 \str_if_in:nnF { #1 } { !! }
5485 {
5486 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

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

```
5487 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5488 }
5489 \int_if_zero:nTF { \l_tmpa_int }
```

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

```
5490 {
5491 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5492 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5493 }
```

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

```
5494 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5495 }
5496 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5497 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

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

```
5498 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5499 {
5500 \dim_compare:nNnT { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim }
5501 {
```

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

```
5502 \group_begin:
5503 \pgfsetcornersarced
5504 {
5505 \pgfpoint
5506 { \l_@@_tab_rounded_corners_dim }
5507 { \l_@@_tab_rounded_corners_dim }
5508 }
```

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.

```
5509 \bool_if:NTF \l_@@_hvlines_bool
5510 {
5511 \pgfpathrectanglecorners
5512 {
5513 \pgfpointadd
5514 { \@@_qpoint:n { row-1 } }
5515 { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5516 }
5517 {
5518 \pgfpointadd
5519 {
```

```

5520         \@@_qpoint:n
5521         { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5522     }
5523     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5524 }
5525 }
5526 {
5527     \pgfpathrectanglecorners
5528     { \@@_qpoint:n { row-1 } }
5529     {
5530         \pgfpointadd
5531         {
5532             \@@_qpoint:n
5533             { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5534         }
5535         { \pgfpoint \c_zero_dim \arrayrulewidth }
5536     }
5537 }
5538 \pgfusepath { clip }
5539 \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5540     }
5541 }

```

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`).

```

5542 \cs_new_protected:Npn \@@_actually_color:
5543 {
5544     \pgfpicture
5545     \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.

```

5546     \@@_clip_with_rounded_corners:
5547     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5548     {
5549         \int_compare:nNnTF { ##1 } = { \c_one_int }
5550         {
5551             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5552             \use:c { g_@@_color _ 1 _tl }
5553             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5554         }
5555         {
5556             \begin { pgfscope }
5557                 \@@_color_opacity: ##2
5558                 \use:c { g_@@_color _ ##1 _tl }
5559                 \tl_gclear:c { g_@@_color _ ##1 _tl }
5560                 \pgfusepath { fill }
5561             \end { pgfscope }
5562         }
5563     }
5564     \endpgfpicture
5565 }

```

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

```

5566 \cs_new_protected:Npn \@@_color_opacity:
5567 {
5568     \peek_meaning:NTF [
5569         { \@@_color_opacity:w }
5570         { \@@_color_opacity:w [ ] }
5571     }

```

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.

```

5572 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5573 {
5574   \tl_clear:N \l_tmpa_tl
5575   \keys_set_known:nnN { nicematrix / color-opacity } { #1 } \l_tmpb_tl
\l_tmpa_tl (if not empty) is now the opacity and \l_tmpb_tl (if not empty) is now the colorimetric
space.

```

```

5576   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5577   \tl_if_empty:NTF \l_tmpb_tl
5578     { \@declaredcolor }
5579     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5580 }

```

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

```

5581 \keys_define:nn { nicematrix / color-opacity }
5582 {
5583   opacity .tl_set:N          = \l_tmpa_tl ,
5584   opacity .value_required:n = true
5585 }

```

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

```

5586 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5587 {
5588   \def \l_@@_rows_tl { #1 }
5589   \def \l_@@_cols_tl { #2 }
5590   \@@_cartesian_path:
5591 }

```

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

```

5592 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5593 {
5594   \tl_if_blank:nF { #2 }
5595   {
5596     \@@_add_to_colors_seq:en
5597     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5598     { \@@_cartesian_color:nn { #3 } { - } }
5599   }
5600 }

```

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

```

5601 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5602 {
5603   \tl_if_blank:nF { #2 }
5604   {
5605     \@@_add_to_colors_seq:en
5606     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5607     { \@@_cartesian_color:nn { - } { #3 } }
5608   }
5609 }

```

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

```

5610 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5611 {
5612   \tl_if_blank:nF { #2 }
5613   {
5614     \@@_add_to_colors_seq:en
5615     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5616     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5617   }
5618 }

```

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

```

5619 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5620 {
5621   \tl_if_blank:nF { #2 }
5622   {
5623     \@@_add_to_colors_seq:en
5624     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5625     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5626   }
5627 }

```

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

```

5628 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5629 {
5630   \@@_cut_on_hyphen:w #1 \q_stop
5631   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5632   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5633   \@@_cut_on_hyphen:w #2 \q_stop
5634   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5635   \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`.

```

5636   \@@_cartesian_path:n { #3 }
5637 }

```

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

```

5638 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5639 {
5640   \clist_map_inline:nn { #3 }
5641   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5642 }

```

```

5643 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5644 {
5645   \int_step_inline:nn { \c@iRow }
5646   {
5647     \int_step_inline:nn { \c@jCol }
5648     {
5649       \int_if_even:nTF { #####1 + ##1 }
5650       { \@@_cellcolor [ #1 ] { #2 } }
5651       { \@@_cellcolor [ #1 ] { #3 } }
5652       { ##1 - #####1 }
5653     }
5654   }
5655 }

```

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”.

```

5656 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5657 {
5658   \@@_rectanglecolor [ #1 ] { #2 }
5659   { 1 - 1 }
5660   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5661 }

5662 \keys_define:nn { nicematrix / rowcolors }
5663 {
5664   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,

```



```

5665     respect-blocks .default:n = true ,
5666     cols .tl_set:N = \l_@@_cols_tl ,
5667     restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5668     restart .default:n = true ,
5669     unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5670 }

```

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*.

```

5671 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5672 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5673     \group_begin:
5674     \seq_clear_new:N \l_@@_colors_seq
5675     \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5676     \tl_clear_new:N \l_@@_cols_tl
5677     \tl_set:Nn \l_@@_cols_tl { - }
5678     \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).

```

5679     \int_zero_new:N \l_@@_color_int
5680     \int_set_eq:NN \l_@@_color_int \c_one_int
5681     \bool_if:NT \l_@@_respect_blocks_bool
5682     {

```

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`).

```

5683         \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5684         \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5685         { \@@_not_in_exterior_p:nnnnn ##1 }
5686     }
5687     \pgfpicture
5688     \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5689     \clist_map_inline:nn { #2 }
5690     {
5691         \tl_set:Nn \l_tmpa_tl { ##1 }
5692         \tl_if_in:NnTF \l_tmpa_tl { - }
5693         { \@@_cut_on_hyphen:w ##1 \q_stop }
5694         { \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.

```

5695         \int_set:Nn \l_tmpa_int \l_tmpa_tl
5696         \int_set:Nn \l_@@_color_int
5697         { \bool_if:NNTF \l_@@_rowcolors_restart_bool { 1 } { \l_tmpa_tl } }
5698         \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5699         \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5700         {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5701         \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).

```

5702     \bool_if:NT \l_@@_respect_blocks_bool
5703     {
5704         \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5705         { \@@_intersect_our_row_p:nnnnn ###1 }
5706         \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5707     }
5708     \tl_set:Ne \l_@@_rows_tl
5709     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5710     \tl_set:Ne \l_@@_color_tl
5711     {
5712         \@@_color_index:n
5713         {
5714             \int_mod:nn
5715             { \l_@@_color_int - 1 }
5716             { \seq_count:N \l_@@_colors_seq }
5717             + 1
5718         }
5719     }
5720     \tl_if_empty:NF \l_@@_color_tl
5721     {
5722         \@@_add_to_colors_seq:ee
5723         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5724         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5725     }
5726     \int_incr:N \l_@@_color_int
5727     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5728 }
5729 }
5730 \endpgfpicture
5731 \group_end:
5732 }

```

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.

```

5733 \cs_new:Npn \@@_color_index:n #1
5734 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5735     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5736     { \@@_color_index:n { #1 - 1 } }
5737     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5738 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by currying.

```

5739 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5740 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around `#3` and `#4` are mandatory.

```

5741 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5742 {
5743     \int_compare:nNnT { #3 } > { \l_tmpb_int }
5744     { \int_set:Nn \l_tmpb_int { #3 } }
5745 }

```

```

5746 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5747 {
5748   \int_if_zero:nTF { #4 }
5749   { \prg_return_false: }
5750   {
5751     \int_compare:nNnTF { #2 } > { \c@jCol }
5752     { \prg_return_false: }
5753     { \prg_return_true: }
5754   }
5755 }

```

The following command return true when the block intersects the row \l_tmpa_int.

```

5756 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5757 {
5758   \int_compare:nNnTF { #1 } > { \l_tmpa_int }
5759   { \prg_return_false: }
5760   {
5761     \int_compare:nNnTF { \l_tmpa_int } > { #3 }
5762     { \prg_return_false: }
5763     { \prg_return_true: }
5764   }
5765 }

```

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).

```

5766 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5767 {
5768   \dim_compare:nNnTF { #1 } = { \c_zero_dim }
5769   {
5770     \bool_if:NTF \l_@@_nocolor_used_bool
5771     { \@@_cartesian_path_normal_ii: }
5772     {
5773       \clist_if_empty:NTF \l_@@_corners_cells_clist
5774       { \@@_cartesian_path_normal_i:n { #1 } }
5775       { \@@_cartesian_path_normal_ii: }
5776     }
5777   }
5778   { \@@_cartesian_path_normal_i:n { #1 } }
5779 }

```

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.

```

5780 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5781 {
5782   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5783   \clist_map_inline:Nn \l_@@_cols_tl
5784   {

```

We use \def instead of \tl_set:Nn for efficiency only.

```

5785     \def \l_tmpa_tl { ##1 }
5786     \tl_if_in:NnTF \l_tmpa_tl { - }
5787     { \@@_cut_on_hyphen:w ##1 \q_stop }
5788     { \def \l_tmpb_tl { ##1 } } % 2025-04-16
5789     \tl_if_empty:NTF \l_tmpa_tl
5790     { \def \l_tmpa_tl { 1 } }
5791     {

```

```

5792     \str_if_eq:eeT \l_tmpa_tl { * }
5793     { \def \l_tmpa_tl { 1 } }
5794 }
5795 \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_col_total_int }
5796 { \@@_error:n { Invalid~col~number } }
5797 \tl_if_empty:NTF \l_tmpb_tl
5798 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5799 {
5800     \str_if_eq:eeT \l_tmpb_tl { * }
5801     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5802 }
5803 \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_col_total_int }
5804 { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5805 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5806 \@@_qpoint:n { col - \l_tmpa_tl }
5807 \int_compare:nNnTF { \l_@@_first_col_int } = { \l_tmpa_tl }
5808 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5809 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5810 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5811 \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.

```

5812 \clist_map_inline:Nn \l_@@_rows_tl
5813 {
5814     \def \l_tmpa_tl { #####1 }
5815     \tl_if_in:NnTF \l_tmpa_tl { - }
5816     { \@@_cut_on_hyphen:w #####1 \q_stop }
5817     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5818     \tl_if_empty:NTF \l_tmpa_tl
5819     { \def \l_tmpa_tl { 1 } }
5820     {
5821         \str_if_eq:eeT \l_tmpa_tl { * }
5822         { \def \l_tmpa_tl { 1 } }
5823     }
5824     \tl_if_empty:NTF \l_tmpb_tl
5825     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5826     {
5827         \str_if_eq:eeT \l_tmpb_tl { * }
5828         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5829     }
5830     \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_row_total_int }
5831     { \@@_error:n { Invalid~row~number } }
5832     \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_row_total_int }
5833     { \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`.

```

5834 \cs_if_exist:cF
5835 { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5836 {
5837     \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5838     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5839     \@@_qpoint:n { row - \l_tmpa_tl }
5840     \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5841     \pgfpathrectanglecorners
5842     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5843     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5844 }
5845 }
5846 }
5847 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5848 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5849 {
5850   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5851   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5852   \clist_map_inline:Nn \l_@@_cols_tl
5853   {
5854     \@@_qpoint:n { col - ##1 }
5855     \int_compare:nNnTF { \l_@@_first_col_int } = { ##1 }
5856     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5857     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5858     \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5859     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5860     \clist_map_inline:Nn \l_@@_rows_tl
5861     {
5862       \@@_if_in_corner:nF { #####1 - ##1 }
5863       {
5864         \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5865         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5866         \@@_qpoint:n { row - #####1 }
5867         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5868         \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5869         {
5870           \pgfpathrectanglecorners
5871           { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5872           { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5873         }
5874       }
5875     }
5876   }
5877 }

```

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`).

```

5878 \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

```

5879 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5880 {
5881   \bool_set_true:N \l_@@_nocolor_used_bool
5882   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5883   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5884   \clist_map_inline:Nn \l_@@_rows_tl
5885   {
5886     \clist_map_inline:Nn \l_@@_cols_tl
5887     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
5888   }
5889 }

```

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.

```

5890 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5891 {
5892   \clist_set_eq:NN \l_tmpa_clist #1

```

```

5893 \clist_clear:N #1
5894 \clist_map_inline:Nn \l_tmpa_clist
5895 {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

5896 \def \l_tmpa_tl { ##1 }
5897 \tl_if_in:NnTF \l_tmpa_tl { - }
5898 { \@@_cut_on_hyphen:w ##1 \q_stop }
5899 { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5900 \bool_lazy_or:nnT
5901 { \str_if_eq_p:ee \l_tmpa_tl { * } }
5902 { \tl_if_blank_p:o \l_tmpa_tl }
5903 { \def \l_tmpa_tl { 1 } }
5904 \bool_lazy_or:nnT
5905 { \str_if_eq_p:ee \l_tmpb_tl { * } }
5906 { \tl_if_blank_p:o \l_tmpb_tl }
5907 { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5908 \int_compare:nNnT { \l_tmpb_tl } > { #2 }
5909 { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5910 \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
5911 { \clist_put_right:Nn #1 { #####1 } }
5912 }
5913 }

```

The following command will be linked to `\cellcolor` in the tabular.

```

5914 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5915 {
5916 \tl_gput_right:Ne \g_@@_pre_code_before_tl
5917 {

```

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`).

```

5918 \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5919 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5920 }
5921 \ignorespaces
5922 }

```

The following command will be linked to `\rowcolor` in the tabular.

```

5923 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5924 {
5925 \tl_gput_right:Ne \g_@@_pre_code_before_tl
5926 {
5927 \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5928 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5929 { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5930 }
5931 \ignorespaces
5932 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5933 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5934 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around `#2` and `#3` are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

5935 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5936 {

```

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`.

```

5937 \seq_gclear:N \g_tmpa_seq
5938 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5939 { \@@_rowlistcolors_tabular:nnnn #1 }
5940 \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`.

```

5941 \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5942 {
5943   { \int_use:N \c@iRow }
5944   { \exp_not:n { #1 } }
5945   { \exp_not:n { #2 } }
5946   { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5947 }
5948 \ignorespaces
5949 }

```

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`).

```

5950 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
5951 {
5952   \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).

```

5953 { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5954 {
5955   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5956   {
5957     \@@_rowlistcolors
5958     [ \exp_not:n { #2 } ]
5959     { #1 - \int_eval:n { \c@iRow - 1 } }
5960     { \exp_not:n { #3 } }
5961     [ \exp_not:n { #4 } ]
5962   }
5963 }
5964 }

```

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.

```

5965 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5966 {
5967   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5968   { \@@_rowlistcolors_tabular_ii:nnnn #1 }
5969   \seq_gclear:N \g_@@_rowlistcolors_seq
5970 }

```

```

5971 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5972 {
5973   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5974   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5975 }

```

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.

```

5976 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5977 {

```

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).

```

5978   \int_compare:nNtT { \c@jCol } > { \g_@@_col_total_int }
5979   {

```

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).

```

5980     \tl_gput_left:Ne \g_@@_pre_code_before_tl
5981     {
5982       \exp_not:N \columncolor [ #1 ]
5983       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5984     }
5985   }
5986 }

```

```

5987 \cs_new_protected:Npn \@@_EmptyColumn:n #1
5988 {
5989   \clist_map_inline:nn { #1 }
5990   {
5991     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
5992     { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
5993     \columncolor { nocolor } { ##1 }
5994   }
5995 }

5996 \cs_new_protected:Npn \@@_EmptyRow:n #1
5997 {
5998   \clist_map_inline:nn { #1 }
5999   {
6000     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6001     { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6002     \rowcolor { nocolor } { ##1 }
6003   }
6004 }

```

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`.

```
6005 \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`.

```
6006 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6007 {
6008   \int_if_zero:nTF { \l_@@_first_col_int }
6009     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6010     {
6011       \int_if_zero:nTF { \c_jCol }
6012       {
6013         \int_compare:nNnF { \c_iRow } = { -1 }
6014         {
6015           \int_compare:nNnF { \c_iRow } = { \l_@@_last_row_int - 1 }
6016           { #1 }
6017         }
6018       }
6019       { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6020     }
6021 }
```

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.

```
6022 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6023 {
6024   \int_if_zero:nF { \c_iRow }
6025   {
6026     \int_compare:nNnF { \c_iRow } = { \l_@@_last_row_int }
6027     {
6028       \int_compare:nNnT { \c_jCol } > { \c_zero_int }
6029       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6030     }
6031   }
6032 }
```

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`.

```
6033 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6034 {
6035   \IfPackageLoadedTF { tikz }
6036   {
6037     \IfPackageLoadedTF { booktabs }
6038     { #2 }
6039     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6040   }
6041   { \@@_error:nn { TopRule~without~tikz } { #1 } }
6042 }

6043 \NewExpandableDocumentCommand { \@@_TopRule } { }
6044 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }

6045 \cs_new:Npn \@@_TopRule_i:
6046 {
6047   \noalign \bgroup
6048   \peek_meaning:NTF [
6049     { \@@_TopRule_ii: }
```

```

6050         { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6051     }
6052 \NewDocumentCommand \@@_TopRule_ii: { o }
6053 {
6054     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6055     {
6056         \@@_hline:n
6057         {
6058             position = \int_eval:n { \c@iRow + 1 } ,
6059             tikz =
6060             {
6061                 line~width = #1 ,
6062                 yshift = 0.25 \arrayrulewidth ,
6063                 shorten~< = - 0.5 \arrayrulewidth
6064             } ,
6065             total~width = #1
6066         }
6067     }
6068     \skip_vertical:n { \belowrulesep + #1 }
6069 \egroup
6070 }
6071 \NewExpandableDocumentCommand { \@@_BottomRule } { } { }
6072 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }
6073 \cs_new:Npn \@@_BottomRule_i:
6074 {
6075     \noalign \bgroup
6076     \peek_meaning:NTF [
6077         { \@@_BottomRule_ii: }
6078         { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6079     }
6080 \NewDocumentCommand \@@_BottomRule_ii: { o }
6081 {
6082     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6083     {
6084         \@@_hline:n
6085         {
6086             position = \int_eval:n { \c@iRow + 1 } ,
6087             tikz =
6088             {
6089                 line~width = #1 ,
6090                 yshift = 0.25 \arrayrulewidth ,
6091                 shorten~< = - 0.5 \arrayrulewidth
6092             } ,
6093             total~width = #1 ,
6094         }
6095     }
6096     \skip_vertical:N \aboverulesep
6097     \@@_create_row_node_i:
6098     \skip_vertical:n { #1 }
6099 \egroup
6100 }
6101 \NewExpandableDocumentCommand { \@@_MidRule } { } { }
6102 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6103 \cs_new:Npn \@@_MidRule_i:
6104 {
6105     \noalign \bgroup
6106     \peek_meaning:NTF [
6107         { \@@_MidRule_ii: }
6108         { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6109     }
6110 \NewDocumentCommand \@@_MidRule_ii: { o }

```

```

6111 {
6112   \skip_vertical:N \aboverulesep
6113   \@@_create_row_node_i:
6114   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6115   {
6116     \@@_hline:n
6117     {
6118       position = \int_eval:n { \c@iRow + 1 } ,
6119       tikz =
6120       {
6121         line-width = #1 ,
6122         yshift = 0.25 \arrayrulewidth ,
6123         shorten~< = - 0.5 \arrayrulewidth
6124       } ,
6125       total-width = #1 ,
6126     }
6127   }
6128   \skip_vertical:n { \belowrulesep + #1 }
6129   \egroup
6130 }

```

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.

```

6131 \keys_define:nn { nicematrix / Rules }
6132 {
6133   position .int_set:N = \l_@@_position_int ,
6134   position .value_required:n = true ,
6135   start .int_set:N = \l_@@_start_int ,
6136   end .code:n =
6137     \bool_lazy_or:nnTF
6138     { \tl_if_empty_p:n { #1 } }
6139     { \str_if_eq_p:ee { #1 } { last } }
6140     { \int_set_eq:NN \l_@@_end_int \c@jCol }
6141     { \int_set:Nn \l_@@_end_int { #1 } }
6142 }

```

It’s possible that the rule won’t be drawn continuously from `start` to `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That’s why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```

6143 \keys_define:nn { nicematrix / RulesBis }
6144 {
6145   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6146   multiplicity .initial:n = 1 ,
6147   dotted .bool_set:N = \l_@@_dotted_bool ,
6148   dotted .initial:n = false ,
6149   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]`.

```

6150   color .code:n =
6151     \@@_set_CTarc:n { #1 }

```

```

6152     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6153     color .value_required:n = true ,
6154     sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6155     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.

```

6156     tikz .code:n =
6157         \IfPackageLoadedTF { tikz }
6158         { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6159         { \@@_error:n { tikz-without-tikz } } ,
6160     tikz .value_required:n = true ,
6161     total-width .dim_set:N = \l_@@_rule_width_dim ,
6162     total-width .value_required:n = true ,
6163     width .meta:n = { total-width = #1 } ,
6164     unknown .code:n = \@@_error:n { Unknown~key~for~RulesBis }
6165 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6166 \cs_new_protected:Npn \@@_vline:n #1
6167 {

```

The group is for the options.

```

6168     \group_begin:
6169     \int_set_eq:NN \l_@@_end_int \c@iRow
6170     \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).

```

6171     \int_compare:nNnT { \l_@@_position_int } < { \c@jCol + 2 }
6172     \@@_vline_i:
6173     \group_end:
6174 }

6175 \cs_new_protected:Npn \@@_vline_i:
6176 {

```

`\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`.

```

6177     \tl_set:Nn \l_tmpb_tl { \int_use:N \l_@@_position_int }
6178     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6179     \l_tmpa_tl
6180     {

```

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.

```

6181     \bool_gset_true:N \g_tmpa_bool
6182     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6183     { \@@_test_vline_in_block:nnnnn ##1 }
6184     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6185     { \@@_test_vline_in_block:nnnnn ##1 }
6186     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6187     { \@@_test_vline_in_stroken_block:nnnn ##1 }
6188     \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_v: }
6189     \bool_if:NTF \g_tmpa_bool
6190     {
6191         \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.

```

6192         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6193     }
6194     {
6195         \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6196         {
6197             \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6198             \@@_vline_ii:
6199             \int_zero:N \l_@@_local_start_int
6200         }
6201     }
6202 }
6203 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6204 {
6205     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6206     \@@_vline_ii:
6207 }
6208 }

6209 \cs_new_protected:Npn \@@_test_in_corner_v:
6210 {
6211     \int_compare:nNnTF { \l_tmpb_tl } = { \c_jCol + 1 }
6212     {
6213         \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6214         { \bool_set_false:N \g_tmpa_bool }
6215     }
6216     {
6217         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6218         {
6219             \int_compare:nNnTF { \l_tmpb_tl } = { \c_one_int }
6220             { \bool_set_false:N \g_tmpa_bool }
6221             {
6222                 \@@_if_in_corner:nT
6223                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6224                 { \bool_set_false:N \g_tmpa_bool }
6225             }
6226         }
6227     }
6228 }

6229 \cs_new_protected:Npn \@@_vline_ii:
6230 {
6231     \tl_clear:N \l_@@_tikz_rule_tl
6232     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6233     \bool_if:NTF \l_@@_dotted_bool
6234     { \@@_vline_iv: }
6235     {
6236         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6237         { \@@_vline_iii: }
6238         { \@@_vline_v: }
6239     }
6240 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

6241 \cs_new_protected:Npn \@@_vline_iii:
6242 {
6243     \pgfpicture
6244     \pgfrememberpicturepositiononpagetrue
6245     \pgf@relevantforpicturesizefalse
6246     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }

```

```

6247 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6248 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6249 \dim_set:Nn \l_tmpb_dim
6250 {
6251   \pgf@x
6252   - 0.5 \l_@@_rule_width_dim
6253   +
6254   ( \arrayrulewidth * \l_@@_multiplicity_int
6255     + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6256 }
6257 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6258 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6259 \bool_lazy_all:nT
6260 {
6261   { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6262   { \cs_if_exist_p:N \CT@drsc@ }
6263   { ! \tl_if_blank_p:o \CT@drsc@ }
6264 }
6265 {
6266   \group_begin:
6267   \CT@drsc@
6268   \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6269   \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6270   \dim_set:Nn \l_@@_tmpd_dim
6271   {
6272     \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6273     * ( \l_@@_multiplicity_int - 1 )
6274   }
6275   \pgfpathrectanglecorners
6276   { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6277   { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6278   \pgfusepath { fill }
6279   \group_end:
6280 }
6281 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6282 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6283 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6284 {
6285   \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6286   \dim_sub:Nn \l_tmpb_dim \doublerulesep
6287   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6288   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6289 }
6290 \CT@arc@
6291 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6292 \pgfsetrectcap
6293 \pgfusepathqstroke
6294 \endpgfpicture
6295 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6296 \cs_new_protected:Npn \@@_vline_iv:
6297 {
6298   \pgfpicture
6299   \pgfrememberpicturepositiononpagetrue
6300   \pgf@relevantforpicturesizefalse
6301   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6302   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6303   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6304   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6305   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6306   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6307   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y

```

```

6308 \CT@arc@
6309 \@@_draw_line:
6310 \endpgfpicture
6311 }

```

The following code is for the case when the user uses the key `tikz`.

```

6312 \cs_new_protected:Npn \@@_vline_v:
6313 {
6314   \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).

```

6315 \CT@arc@
6316 \tl_if_empty:NF \l_@@_rule_color_tl
6317 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6318 \pgfrememberpicturepositiononpagetrue
6319 \pgf@relevantforpicturesizefalse
6320 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6321 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6322 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6323 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6324 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6325 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6326 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6327 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6328 ( \l_tmpb_dim , \l_tmpa_dim ) --
6329 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6330 \end { tikzpicture }
6331 }

```

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).

```

6332 \cs_new_protected:Npn \@@_draw_vlines:
6333 {
6334   \int_step_inline:nnn
6335   {
6336     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6337     { 2 }
6338     { 1 }
6339   }
6340   {
6341     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6342     { \c@jCol }
6343     { \int_eval:n { \c@jCol + 1 } }
6344   }
6345   {
6346     \str_if_eq:eeF \l_@@_vlines_clist { all }
6347     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6348     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6349   }
6350 }

```

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}`.

```

6351 \cs_new_protected:Npn \@@_hline:n #1
6352 {

```

The group is for the options.

```

6353 \group_begin:
6354 \int_set_eq:NN \l_@@_end_int \c@jCol
6355 \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6356 \@@_hline_i:
6357 \group_end:
6358 }

6359 \cs_new_protected:Npn \@@_hline_i:
6360 {
6361   % \int_zero:N \l_@@_local_start_int
6362   % \int_zero:N \l_@@_local_end_int

```

\l_tmpa_tl is the number of row and \l_tmpb_tl the number of column. When we have found a column corresponding to a rule to draw, we note its number in $\l_@@_tmpc_tl$.

```

6363 \tl_set:N \l_@@_tmpc_tl { \int_use:N \l_@@_position_int }
6364 \int_step_variable:nnN \l_@@_start_int \l_@@_end_int
6365   \l_@@_tmpb_tl
6366   {

```

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 $\backslash Block$ or a virtual block corresponding to a dotted line, created by $\backslash Cdots$, $\backslash Vdots$, etc.), we will set \g_tmpa_bool to false and the small horizontal rule won't be drawn.

```

6367 \bool_gset_true:N \g\_tmpa\_bool

```

We test whether we are in a block.

```

6368 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6369   { \@@_test_hline_in_block:nnnnn ##1 }

6370 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6371   { \@@_test_hline_in_block:nnnnn ##1 }
6372 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6373   { \@@_test_hline_in_stroken_block:nnnn ##1 }
6374 \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_h: }
6375 \bool_if:NTF \g\_tmpa\_bool
6376   {
6377     \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.

```

6378   { \int_set:Nn \l_@@_local_start_int \l_@@_local_start_int }
6379   }
6380   {
6381     \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6382     {
6383       \int_set:Nn \l_@@_local_end_int { \l_@@_local_start_int - 1 }
6384       \@@_hline_ii:
6385       \int_zero:N \l_@@_local_start_int
6386     }
6387   }
6388 }
6389 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6390 {
6391   \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6392   \@@_hline_ii:
6393 }
6394 }

```

```

6395 \cs_new_protected:Npn \@@_test_in_corner_h:
6396 {
6397   \int_compare:nNnTF { \l_@@_local_start_int } = { \c@iRow + 1 }
6398   {
6399     \@@_if_in_corner:nT { \int_eval:n { \l_@@_local_start_int - 1 } - \l_@@_local_end_int }

```



```

6400         { \bool_set_false:N \g_tmpa_bool }
6401     }
6402     {
6403         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6404         {
6405             \int_compare:nNnTF { \l_tmpa_tl } = { \c_one_int }
6406             { \bool_set_false:N \g_tmpa_bool }
6407             {
6408                 \@@_if_in_corner:nT
6409                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6410                 { \bool_set_false:N \g_tmpa_bool }
6411             }
6412         }
6413     }
6414 }

```

```

6415 \cs_new_protected:Npn \@@_hline_ii:
6416 {
6417     \tl_clear:N \l_@@_tikz_rule_tl
6418     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6419     \bool_if:NTF \l_@@_dotted_bool
6420     { \@@_hline_iv: }
6421     {
6422         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6423         { \@@_hline_iii: }
6424         { \@@_hline_v: }
6425     }
6426 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6427 \cs_new_protected:Npn \@@_hline_iii:
6428 {
6429     \pgfpicture
6430     \pgfrememberpicturepositiononpagetrue
6431     \pgf@relevantforpicturesizefalse
6432     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6433     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6434     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6435     \dim_set:Nn \l_tmpb_dim
6436     {
6437         \pgf@y
6438         - 0.5 \l_@@_rule_width_dim
6439         +
6440         ( \arrayrulewidth * \l_@@_multiplicity_int
6441           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6442     }
6443     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6444     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6445     \bool_lazy_all:nT
6446     {
6447         { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6448         { \cs_if_exist_p:N \CT@drsc@ }
6449         { ! \tl_if_blank_p:o \CT@drsc@ }
6450     }
6451     {
6452         \group_begin:
6453         \CT@drsc@
6454         \dim_set:Nn \l_@@_tmpd_dim
6455         {
6456             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6457             * ( \l_@@_multiplicity_int - 1 )
6458         }

```

```

6459     \pgfpathrectanglecorners
6460     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6461     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6462     \pgfusepathqfill
6463     \group_end:
6464 }
6465 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6466 \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6467 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6468 {
6469     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6470     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6471     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6472     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6473 }
6474 \CT@arc@
6475 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6476 \pgfsetrectcap
6477 \pgfusepathqstroke
6478 \endpgfpicture
6479 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6480 \cs_new_protected:Npn \@@_hline_iv:
6481 {
6482     \pgfpicture
6483     \pgfrememberpicturepositiononpagetrue
6484     \pgf@relevantforpicturesizefalse
6485     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6486     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6487     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6488     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6489     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6490     \int_compare:nNnT { \l_@@_local_start_int } = { \c_one_int }
6491     {
6492         \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6493         \bool_if:NF \g_@@_delims_bool
6494         { \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.

```

6495     \tl_if_eq:NnF \g_@@_left_delim_tl (
6496     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6497     )
6498     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6499     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x

```

```

6500 \int_compare:nNtT { \l_@@_local_end_int } = { \c@jCol }
6501 {
6502   \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6503   \bool_if:NF \g_@@_delims_bool
6504   { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6505   \tl_if_eq:NnF \g_@@_right_delim_tl )
6506   { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6507 }
6508 \CT@arc@
6509 \@@_draw_line:
6510 \endpgfpicture
6511 }

```

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`).

```

6512 \cs_new_protected:Npn \@@_hline_v:
6513 {
6514   \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).

```

6515   \CT@arc@
6516   \tl_if_empty:NF \l_@@_rule_color_tl
6517   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6518   \pgfrememberpicturepositiononpagetrue
6519   \pgf@relevantforpicturesizefalse
6520   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6521   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6522   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6523   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6524   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6525   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6526   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6527   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6528   ( \l_tmpa_dim , \l_tmpb_dim ) --
6529   ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6530   \end { tikzpicture }
6531 }

```

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).

```

6532 \cs_new_protected:Npn \@@_draw_hlines:
6533 {
6534   \int_step_inline:nnn
6535   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6536   {
6537     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6538     { \c@iRow }
6539     { \int_eval:n { \c@iRow + 1 } } }
6540   }
6541   {
6542     \str_if_eq:eeF \l_@@_hlines_clist { all }
6543     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6544     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6545   }
6546 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6547 \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.

```

6548 \cs_set:Npn \@@_Hline_i:n #1
6549 {
6550   \peek_remove_spaces:n
6551   {
6552     \peek_meaning:NTF \Hline
6553     { \@@_Hline_ii:nn { #1 + 1 } }
6554     { \@@_Hline_iii:n { #1 } }
6555   }
6556 }
6557 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6558 \cs_set:Npn \@@_Hline_iii:n #1
6559 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6560 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6561 {
6562   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6563   \skip_vertical:N \l_@@_rule_width_dim
6564   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6565   {
6566     \@@_hline:n
6567     {
6568       multiplicity = #1 ,
6569       position = \int_eval:n { \c@iRow + 1 } ,
6570       total-width = \dim_use:N \l_@@_rule_width_dim ,
6571       #2
6572     }
6573   }
6574   \egroup
6575 }

```

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`).

```

6576 \cs_new_protected:Npn \@@_custom_line:n #1
6577 {
6578   \str_clear_new:N \l_@@_command_str
6579   \str_clear_new:N \l_@@_ccommand_str
6580   \str_clear_new:N \l_@@_letter_str
6581   \tl_clear_new:N \l_@@_other_keys_tl
6582   \keys_set:known:n { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6583   \bool_lazy_all:nTF
6584   {
6585     { \str_if_empty_p:N \l_@@_letter_str }
6586     { \str_if_empty_p:N \l_@@_command_str }
6587     { \str_if_empty_p:N \l_@@_ccommand_str }
6588   }
6589   { \@@_error:n { No~letter~and~no~command } }
6590   { \@@_custom_line_i:o \l_@@_other_keys_tl }
6591 }
6592 \keys_define:nn { nicematrix / custom-line }
6593 {
6594   letter .str_set:N = \l_@@_letter_str ,

```

```

6595     letter .value_required:n = true ,
6596     command .str_set:N = \l_@@_command_str ,
6597     command .value_required:n = true ,
6598     ccommand .str_set:N = \l_@@_ccommand_str ,
6599     ccommand .value_required:n = true ,
6600 }

```

```

6601 \cs_new_protected:Npn \@@_custom_line_i:n #1
6602 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6603     \bool_set_false:N \l_@@_tikz_rule_bool
6604     \bool_set_false:N \l_@@_dotted_rule_bool
6605     \bool_set_false:N \l_@@_color_bool
6606     \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6607     \bool_if:NT \l_@@_tikz_rule_bool
6608     {
6609         \IfPackageLoadedF { tikz }
6610         { \@@_error:n { tikz~in~custom-line-without~tikz } }
6611         \bool_if:NT \l_@@_color_bool
6612         { \@@_error:n { color~in~custom-line-with~tikz } }
6613     }
6614     \bool_if:NT \l_@@_dotted_rule_bool
6615     {
6616         \int_compare:nNnT { \l_@@_multiplicity_int } > { \c_one_int }
6617         { \@@_error:n { key~multiplicity~with~dotted } }
6618     }
6619     \str_if_empty:NF \l_@@_letter_str
6620     {
6621         \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6622         { \@@_error:n { Several~letters } }
6623         {
6624             \tl_if_in:NoTF
6625             \c_@@_forbidden_letters_str
6626             \l_@@_letter_str
6627             { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6628         }

```

During the analyse of the preamble provided by the final user, our automaton, for the letter corresponding at the custom line, will directly use the following command that you define in the main hash table of TeX.

```

6629         \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6630         { \@@_v_custom_line:n { #1 } }
6631     }
6632 }
6633 }
6634 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6635 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6636 }
6637 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6638 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6639 \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.

```

6640 \keys_define:nn { nicematrix / custom-line-bis }
6641 {
6642     multiplicity .int_set:N = \l_@@_multiplicity_int ,

```

```

6643     multiplicity .initial:n = 1 ,
6644     multiplicity .value_required:n = true ,
6645     color .code:n = \bool_set_true:N \l_@@_color_bool ,
6646     color .value_required:n = true ,
6647     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6648     tikz .value_required:n = true ,
6649     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6650     dotted .value_forbidden:n = true ,
6651     total-width .code:n = { } ,
6652     total-width .value_required:n = true ,
6653     width .code:n = { } ,
6654     width .value_required:n = true ,
6655     sep-color .code:n = { } ,
6656     sep-color .value_required:n = true ,
6657     unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
6658 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6659 \bool_new:N \l_@@_dotted_rule_bool
6660 \bool_new:N \l_@@_tikz_rule_bool
6661 \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`.

```

6662 \keys_define:nn { nicematrix / custom-line-width }
6663 {
6664     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6665     multiplicity .initial:n = 1 ,
6666     multiplicity .value_required:n = true ,
6667     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6668     total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6669                     \bool_set_true:N \l_@@_total_width_bool ,
6670     total-width .value_required:n = true ,
6671     width .meta:n = { total-width = #1 } ,
6672     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6673 }

```

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`).

```

6674 \cs_new_protected:Npn \@@_h_custom_line:n #1
6675 {

```

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`).

```

6676     \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6677     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6678 }

```

The following command will create the command that the final user will use in its array to draw an horizontal line 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`).

```

6679 \cs_new_protected:Npn \@@_c_custom_line:n #1
6680 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

6681     \exp_args:Nc \NewExpandableDocumentCommand
6682     { nicematrix - \l_@@_ccommand_str }
6683     { 0 { } m }

```

```

6684 {
6685   \noalign
6686   {
6687     \@@_compute_rule_width:n { #1 , ##1 }
6688     \skip_vertical:n { \l_@@_rule_width_dim }
6689     \clist_map_inline:nn
6690       { ##2 }
6691       { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6692   }
6693 }
6694 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6695 }

```

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*.

```

6696 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6697 {
6698   \tl_if_in:nnTF { #2 } { - }
6699   { \@@_cut_on_hyphen:w #2 \q_stop }
6700   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6701   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6702   {
6703     \@@_hline:n
6704     {
6705       #1 ,
6706       start = \l_tmpa_tl ,
6707       end = \l_tmpb_tl ,
6708       position = \int_eval:n { \c@iRow + 1 } ,
6709       total-width = \dim_use:N \l_@@_rule_width_dim
6710     }
6711   }
6712 }
6713 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6714 {
6715   \bool_set_false:N \l_@@_tikz_rule_bool
6716   \bool_set_false:N \l_@@_total_width_bool
6717   \bool_set_false:N \l_@@_dotted_rule_bool
6718   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6719   \bool_if:NF \l_@@_total_width_bool
6720   {
6721     \bool_if:NTF \l_@@_dotted_rule_bool
6722     { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6723     {
6724       \bool_if:NF \l_@@_tikz_rule_bool
6725       {
6726         \dim_set:Nn \l_@@_rule_width_dim
6727         {
6728           \arrayrulewidth * \l_@@_multiplicity_int
6729           + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6730         }
6731       }
6732     }
6733   }
6734 }
6735 \cs_new_protected:Npn \@@_v_custom_line:n #1
6736 {
6737   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6738   \tl_gput_right:Ne \g_@@_array_preamble_tl
6739   { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6740   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6741   {

```

```

6742 \@@_vline:n
6743 {
6744   #1 ,
6745   position = \int_eval:n { \c@jCol + 1 } ,
6746   total-width = \dim_use:N \l_@@_rule_width_dim
6747 }
6748 }
6749 \@@_rec_preamble:n
6750 }
6751 \@@_custom_line:n
6752 { 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.

```

6753 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6754 {
6755   \int_compare:nNnT { \l_tmpa_tl } > { #1 }
6756   {
6757     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6758     {
6759       \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6760       {
6761         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6762         { \bool_gset_false:N \g_tmpa_bool }
6763       }
6764     }
6765   }
6766 }

```

The same for vertical rules.

```

6767 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6768 {
6769   \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6770   {
6771     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6772     {
6773       \int_compare:nNnT { \l_tmpb_tl } > { #2 }
6774       {
6775         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6776         { \bool_gset_false:N \g_tmpa_bool }
6777       }
6778     }
6779   }
6780 }
6781 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6782 {
6783   \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6784   {
6785     \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6786     {
6787       \int_compare:nNnTF { \l_tmpa_tl } = { #1 }
6788       { \bool_gset_false:N \g_tmpa_bool }
6789       {
6790         \int_compare:nNnT { \l_tmpa_tl } = { #3 + 1 }
6791         { \bool_gset_false:N \g_tmpa_bool }
6792       }
6793     }
6794   }
6795 }

```



```

6796 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6797 {
6798   \int_compare:nNtT { \l_tmpa_tl } > { #1 - 1 }
6799   {
6800     \int_compare:nNtT { \l_tmpa_tl } < { #3 + 1 }
6801     {
6802       \int_compare:nNtTF { \l_tmpb_tl } = { #2 }
6803       { \bool_gset_false:N \g_tmpa_bool }
6804       {
6805         \int_compare:nNtT { \l_tmpb_tl } = { #4 + 1 }
6806         { \bool_gset_false:N \g_tmpa_bool }
6807       }
6808     }
6809   }
6810 }

```

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.

```

6811 \cs_new_protected:Npn \@@_compute_corners:
6812 {
6813   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6814   { \@@_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`).

```

6815   \clist_clear:N \l_@@_corners_cells_clist
6816   \clist_map_inline:Nn \l_@@_corners_cells_clist
6817   {
6818     \str_case:nnF { ##1 }
6819     {
6820       { NW }
6821       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6822       { NE }
6823       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6824       { SW }
6825       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6826       { SE }
6827       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6828     }
6829     { \@@_error:nn { bad~corner } { ##1 } }
6830   }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6831   \clist_if_empty:NF \l_@@_corners_cells_clist
6832   {

```

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.

```

6833     \tl_gput_right:Ne \g_@@_aux_tl
6834     {
6835       \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
6836       { \l_@@_corners_cells_clist }
6837     }
6838   }
6839 }

```

```

6840 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6841 {
6842   \int_step_inline:nnn { #1 } { #3 }
6843   {
6844     \int_step_inline:nnn { #2 } { #4 }
6845     { \cs_set_nopar:cpn { @@_block_ ##1 - ###1 } { } }
6846   }
6847 }

6848 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6849 {
6850   \cs_if_exist:cTF
6851   { @@_block_ \int_eval:n { #1 } - \int_eval:n { #2 } }
6852   { \prg_return_true: }
6853   { \prg_return_false: }
6854 }

```

“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.

```

6855 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6856 {

```

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.

```

6857   \bool_set_false:N \l_tmpa_bool
6858   \int_zero_new:N \l_@@_last_empty_row_int
6859   \int_set:Nn \l_@@_last_empty_row_int { #1 }
6860   \int_step_inline:nnnn { #1 } { #3 } { #5 }
6861   {
6862     \bool_lazy_or:nnTF
6863     {
6864       \cs_if_exist_p:c
6865       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6866     }
6867     { \@@_if_in_block_p:nn { ##1 } { #2 } }
6868     { \bool_set_true:N \l_tmpa_bool }
6869     {
6870       \bool_if:NF \l_tmpa_bool
6871       { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6872     }
6873   }

```

Now, you determine the last empty cell in the row of number 1.

```

6874   \bool_set_false:N \l_tmpa_bool
6875   \int_zero_new:N \l_@@_last_empty_column_int
6876   \int_set:Nn \l_@@_last_empty_column_int { #2 }
6877   \int_step_inline:nnnn { #2 } { #4 } { #6 }
6878   {
6879     \bool_lazy_or:nnTF
6880     {

```

```

6881         \cs_if_exist_p:c
6882         { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6883     }
6884     { \@@_if_in_block_p:nn { #1 } { ##1 } }
6885     { \bool_set_true:N \l_tmpa_bool }
6886     {
6887         \bool_if:NF \l_tmpa_bool
6888         { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6889     }
6890 }

```

Now, we loop over the rows.

```

6891     \int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int }
6892     {

```

We treat the row number ##1 with another loop.

```

6893         \bool_set_false:N \l_tmpa_bool
6894         \int_step_inline:nnnn { #2 } { #4 } { \l_@@_last_empty_column_int }
6895         {
6896             \bool_lazy_or:nnTF
6897             { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6898             { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6899             { \bool_set_true:N \l_tmpa_bool }
6900             {
6901                 \bool_if:NF \l_tmpa_bool
6902                 {
6903                     \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6904                     \clist_put_right:Nn
6905                     \l_@@_corners_cells_clist
6906                     { ##1 - #####1 }
6907                     \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6908                 }
6909             }
6910         }
6911     }
6912 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6913 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6914 \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.

```

6915 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

6916 \keys_define:nn { nicematrix / NiceMatrixBlock }
6917 {
6918     auto-columns-width .code:n =
6919     {
6920         \bool_set_true:N \l_@@_block_auto_columns_width_bool
6921         \dim_gzero_new:N \g_@@_max_cell_width_dim
6922         \bool_set_true:N \l_@@_auto_columns_width_bool
6923     }
6924 }

```

```

6925 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6926 {
6927   \int_gincr:N \g_@@_NiceMatrixBlock_int
6928   \dim_zero:N \l_@@_columns_width_dim
6929   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6930   \bool_if:NT \l_@@_block_auto_columns_width_bool
6931   {
6932     \cs_if_exist:cT
6933     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6934     {
6935       \dim_set:Nn \l_@@_columns_width_dim
6936       {
6937         \use:c
6938         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6939       }
6940     }
6941   }
6942 }

```

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`).

```

6943 {
6944   \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.

```

6945   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6946   {
6947     \bool_if:NT \l_@@_block_auto_columns_width_bool
6948     {
6949       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6950       \iow_shipout:Ne \@mainaux
6951       {
6952         \cs_gset:cpn
6953         { @@ _ 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.

```

6954         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6955       }
6956       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6957     }
6958   }
6959   \ignorespacesafterend
6960 }

```

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).

```

6961 \cs_new_protected:Npn \@@_create_extra_nodes:
6962 {
6963   \bool_if:nTF \l_@@_medium_nodes_bool
6964   {
6965     \bool_if:NTF \l_@@_no_cell_nodes_bool
6966     { \@@_error:n { extra-nodes~with~no~cell~nodes } }
6967     {
6968       \bool_if:NTF \l_@@_large_nodes_bool

```

```

6969         \@@_create_medium_and_large_nodes:
6970         \@@_create_medium_nodes:
6971     }
6972 }
6973 {
6974     \bool_if:NT \l_@@_large_nodes_bool
6975     {
6976         \bool_if:NTF \l_@@_no_cell_nodes_bool
6977         { \@@_error:n { extra-nodes-with-no-cell-nodes } }
6978         \@@_create_large_nodes:
6979     }
6980 }
6981 }

```

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`.

```

6982 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6983 {
6984     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6985     {
6986         \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
6987         \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
6988         \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
6989         \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
6990     }
6991     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6992     {
6993         \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
6994         \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
6995         \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
6996         \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
6997     }

```

We begin the two nested loops over the rows and the columns of the array.

```

6998     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6999     {
7000         \int_step_variable:nnNn
7001         \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.

```

7002     {
7003         \cs_if_exist:cT
7004         { 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`.

```

7005     {
7006         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7007         \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7008         { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7009         \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7010         {
7011             \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7012             { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7013         }

```

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`.

```

7014         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7015         \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7016         { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } { \pgf@y } }
7017         \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7018         {
7019             \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7020             { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } { \pgf@x } }
7021         }
7022     }
7023 }
7024 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7025 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7026 {
7027     \dim_compare:nNnT
7028     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7029     {
7030         \@@_qpoint:n { row - \@@_i: - base }
7031         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7032         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7033     }
7034 }
7035 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7036 {
7037     \dim_compare:nNnT
7038     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7039     {
7040         \@@_qpoint:n { col - \@@_j: }
7041         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7042         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7043     }
7044 }
7045 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7046 \cs_new_protected:Npn \@@_create_medium_nodes:
7047 {
7048     \pgfpicture
7049     \pgfrememberpicturepositiononpagetrue
7050     \pgf@relevantforpicturesizefalse
7051     \@@_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”.

```

7052     \tl_set:Nn \l_@@_suffix_tl { -medium }
7053     \@@_create_nodes:

```

```

7054 \endpgfpicture
7055 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones¹⁵. 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:`.

```

7056 \cs_new_protected:Npn \@@_create_large_nodes:
7057 {
7058   \pgfpicture
7059   \pgfrememberpicturepositiononpagetrue
7060   \pgf@relevantforpicturesizefalse
7061   \@@_computations_for_medium_nodes:
7062   \@@_computations_for_large_nodes:
7063   \tl_set:Nn \l_@@_suffix_tl { - large }
7064   \@@_create_nodes:
7065   \endpgfpicture
7066 }
7067 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7068 {
7069   \pgfpicture
7070   \pgfrememberpicturepositiononpagetrue
7071   \pgf@relevantforpicturesizefalse
7072   \@@_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”.

```

7073   \tl_set:Nn \l_@@_suffix_tl { - medium }
7074   \@@_create_nodes:
7075   \@@_computations_for_large_nodes:
7076   \tl_set:Nn \l_@@_suffix_tl { - large }
7077   \@@_create_nodes:
7078   \endpgfpicture
7079 }

```

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.

```

7080 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7081 {
7082   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7083   \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`.

```

7084   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7085   {
7086     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7087     {
7088       (
7089         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7090         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7091       )
7092       / 2
7093     }
7094     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7095     { l_@@_row _ \@@_i: _ min _ dim }
7096   }
7097   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:

```

¹⁵If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7098 {
7099   \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7100   {
7101     (
7102       \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7103       \dim_use:c
7104         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7105     )
7106     / 2
7107   }
7108   \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7109   { l_@@_column _ \@@_j: _ max _ dim }
7110 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7111 \dim_sub:cn
7112   { l_@@_column _ 1 _ min _ dim }
7113   \l_@@_left_margin_dim
7114 \dim_add:cn
7115   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7116   \l_@@_right_margin_dim
7117 }

```

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).

```

7118 \cs_new_protected:Npn \@@_create_nodes:
7119 {
7120   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7121   {
7122     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7123     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7124       \@@_pgf_rect_node:nnnnn
7125       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7126       { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7127       { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7128       { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7129       { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7130       \str_if_empty:NF \l_@@_name_str
7131       {
7132         \pgfnodealias
7133         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7134         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7135       }
7136     }
7137   }
7138   \int_step_inline:nn { \c@iRow }
7139   {
7140     \pgfnodealias
7141     { \@@_env: - ##1 - last \l_@@_suffix_tl }
7142     { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7143   }
7144   \int_step_inline:nn { \c@jCol }
7145   {
7146     \pgfnodealias
7147     { \@@_env: - last - ##1 \l_@@_suffix_tl }
7148     { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7149   }

```



```

7150 \pgfnodealias % added 2025-04-05
7151 { \@@_env: - last - last \l_@@_suffix_tl }
7152 { \@@_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 .

```

7153 \seq_map_pairwise_function:NNN
7154 \g_@@_multicolumn_cells_seq
7155 \g_@@_multicolumn_sizes_seq
7156 \@@_node_for_multicolumn:nn
7157 }

7158 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7159 {
7160   \cs_set_nopar:Npn \@@_i: { #1 }
7161   \cs_set_nopar:Npn \@@_j: { #2 }
7162 }

```

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”).

```

7163 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7164 {
7165   \@@_extract_coords_values: #1 \q_stop
7166   \@@_pgf_rect_node:nnnnn
7167   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7168   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7169   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7170   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: + #2 - 1 } _ max _ dim } }
7171   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7172   \str_if_empty:NF \l_@@_name_str
7173   {
7174     \pgfnodealias
7175     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7176     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7177   }
7178 }

```

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).

```

7179 \keys_define:nn { nicematrix / Block / FirstPass }
7180 {
7181   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7182   \bool_set_true:N \l_@@_p_block_bool ,
7183   j .value_forbidden:n = true ,
7184   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7185   l .value_forbidden:n = true ,
7186   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7187   r .value_forbidden:n = true ,
7188   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7189   c .value_forbidden:n = true ,

```

```

7190 L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7191 L .value_forbidden:n = true ,
7192 R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7193 R .value_forbidden:n = true ,
7194 C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7195 C .value_forbidden:n = true ,
7196 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7197 t .value_forbidden:n = true ,
7198 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7199 T .value_forbidden:n = true ,
7200 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7201 b .value_forbidden:n = true ,
7202 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7203 B .value_forbidden:n = true ,
7204 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7205 m .value_forbidden:n = true ,
7206 v-center .meta:n = m ,
7207 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7208 p .value_forbidden:n = true ,
7209 color .code:n =
7210   \@@_color:n { #1 }
7211   \tl_set_rescan:Nnn
7212     \l_@@_draw_tl
7213     { \char_set_catcode_other:N ! }
7214     { #1 } ,
7215 color .value_required:n = true ,
7216 respect-arraystretch .code:n =
7217   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7218 respect-arraystretch .value_forbidden:n = true ,
7219 }

```

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.

```

7220 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7221 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7222 {

```

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).

```

7223   \tl_if_blank:nTF { #2 }
7224   { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7225   {
7226     \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7227     \@@_Block_i_czech:w \@@_Block_i:w
7228     #2 \q_stop
7229   }
7230   { #1 } { #3 } { #4 }
7231   \ignorespaces
7232 }

```

With the following construction, we extract the values of *i* and *j* in the first mandatory argument of the command.

```

7233 \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`.

```

7234 {
7235   \char_set_catcode_active:N -
7236   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7237 }

```

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.

```
7238 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7239 {
```

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).

```
7240 \bool_lazy_or:nnTF
7241 { \tl_if_blank_p:n { #1 } }
7242 { \str_if_eq_p:ee { * } { #1 } }
7243 { \int_set:Nn \l_tmpa_int { 100 } }
7244 { \int_set:Nn \l_tmpa_int { #1 } }
7245 \bool_lazy_or:nnTF
7246 { \tl_if_blank_p:n { #2 } }
7247 { \str_if_eq_p:ee { * } { #2 } }
7248 { \int_set:Nn \l_tmpb_int { 100 } }
7249 { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```
7250 \int_compare:nNnTF { \l_tmpb_int } = { \c_one_int }
7251 {
7252   \tl_if_empty:NTF \l_@@_hpos_cell_tl
7253   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7254   { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7255 }
7256 { \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.

```
7257 \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7258 \tl_set:Ne \l_tmpa_tl
7259 {
7260   { \int_use:N \c@iRow }
7261   { \int_use:N \c@jCol }
7262   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7263   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7264 }
```

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).

```
7265 \bool_set_false:N \l_tmpa_bool
7266 \bool_if:NT \l_@@_amp_in_blocks_bool
```

\tl_if_in:nnT is slightly faster than \str_if_in:nnT.

```
7267 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7268 \bool_case:nF
7269 {
7270   \l_tmpa_bool { \@@_Block_vii:eennn }
7271   \l_@@_p_block_bool { \@@_Block_vi:eennn }
```

For the blocks mono-column, we will compose right now in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a X column, we should not do that since the width is determined by another way. This should be the same for the p, m and b columns and we should modify that point. However, for the X column, it's imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

7272         \l_@@_X_bool                                { \@@_Block_v:eennn }
7273         { \tl_if_empty_p:n { #5 } }                  { \@@_Block_v:eennn }
7274         { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7275         { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7276     }
7277     { \@@_Block_v:eennn }
7278     { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7279 }

```

The following macro is for the case of a `\Block` which is mono-row or mono-column (or both) and don't use the key p. In that case, the content of the block is composed right now in a box (because we have to take into account the dimensions of that box for the width of the current column or the height and the depth of the current row). However, that box will be put in the array *after the construction of the array* (by using PGF) with `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn` which will do the main job.

#1 is *i* (the number of rows of the block), #2 is *j* (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the potential math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7280 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7281 {
7282     \int_gincr:N \g_@@_block_box_int
7283     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7284     {
7285         \tl_gput_right:Ne \g_@@_pre_code_after_tl
7286         {
7287             \@@_actually_diagbox:nnnnnn
7288             { \int_use:N \c@iRow }
7289             { \int_use:N \c@jCol }
7290             { \int_eval:n { \c@iRow + #1 - 1 } }
7291             { \int_eval:n { \c@jCol + #2 - 1 } }
7292             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7293             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7294         }
7295     }
7296     \box_gclear_new:c
7297     { g_@@_block_box_int _box _ \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!

```

7298     \hbox_gset:cn
7299     { g_@@_block_box_int _box _ \int_use:N \g_@@_block_box_int _box }
7300     {

```

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`).

```

7301         \tl_if_empty:NTF \l_@@_color_tl
7302         { \int_compare:nNnT { #2 } = { \c_one_int } { \set@color } }
7303         { \@@_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`.

```

7304         \int_compare:nNnT { #1 } = { \c_one_int }

```

```

7305         {
7306         \int_if_zero:nTF { \c@iRow }
7307         {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7308         \cs_set_eq:NN \Block \@@_NullBlock:
7309         \l_@@_code_for_first_row_tl
7310     }
7311     {
7312         \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7313         {
7314             \cs_set_eq:NN \Block \@@_NullBlock:
7315             \l_@@_code_for_last_row_tl
7316         }
7317     }
7318     \g_@@_row_style_tl
7319 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7320 \@@_reset_arraystretch:
7321 \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7322 #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.).

```

7323 \@@_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}`.

```

7324 \bool_if:NTF \l_@@_tabular_bool
7325 {
7326     \bool_lazy_all:nTF
7327     {
7328         { \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.

```

7329         {
7330             ! \dim_compare_p:nNn
7331               { \l_@@_col_width_dim } < { \c_zero_dim }
7332         }
7333         { ! \g_@@_rotate_bool }
7334     }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7335         {
7336             \use:e
7337         {

```

The `\exp_not:N` is mandatory before `\begin`. It will be possible to delete the `\exp_not:N` in TeXLive 2025 because `\begin` is now protected by `\protected` (and not by `\protect`). There is several other occurrences in that document.

```

7338             \exp_not:N \begin { minipage }
7339             [ \str_lowercase:f \l_@@_vpos_block_str ]
7340             { \l_@@_col_width_dim }
7341             \str_case:on \l_@@_hpos_block_str
7342               { c \centering r \raggedleft l \raggedright }
7343         }
7344         #5
7345     \end { minipage }
7346 }

```

In the other cases, we use a `{tabular}`.

```

7347         {
7348             \bool_if:NT \c_@@_testphase_table_bool
7349             { \tagpdfsetup { table / tagging = presentation } }
7350             \use:e
7351             {
7352                 \exp_not:N \begin { tabular }
7353                 [ \str_lowercase:f \l_@@_vpos_block_str ]
7354                 { @ { } \l_@@_hpos_block_str @ { } }
7355             }
7356             #5
7357         \end { tabular }
7358     }
7359 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7360         {
7361             \c_math_toggle_token
7362             \use:e
7363             {
7364                 \exp_not:N \begin { array }
7365                 [ \str_lowercase:f \l_@@_vpos_block_str ]
7366                 { @ { } \l_@@_hpos_block_str @ { } }
7367             }
7368             #5
7369         \end { array }
7370             \c_math_toggle_token
7371     }
7372 }

```

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).

```

7373     \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.

```

7374     \int_compare:nNnT { #2 } = { \c_one_int }
7375     {
7376         \dim_gset:Nn \g_@@_blocks_wd_dim
7377         {
7378             \dim_max:nn
7379             { \g_@@_blocks_wd_dim }
7380             {
7381                 \box_wd:c
7382                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7383             }
7384         }
7385     }

```

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.

```

7386     \bool_lazy_and:nnT
7387     { \int_compare_p:nNn { #1 } = { \c_one_int } }

```

If the user has not used a key for the vertical position of the block, then `\l_@@_vpos_block_str` remains empty.

```

7388     { \str_if_empty_p:N \l_@@_vpos_block_str }
7389     {
7390         \dim_gset:Nn \g_@@_blocks_ht_dim
7391         {
7392             \dim_max:nn
7393             { \g_@@_blocks_ht_dim }
7394             {
7395                 \box_ht:c
7396                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7397             }
7398         }
7399         \dim_gset:Nn \g_@@_blocks_dp_dim
7400         {
7401             \dim_max:nn
7402             { \g_@@_blocks_dp_dim }
7403             {
7404                 \box_dp:c
7405                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7406             }
7407         }
7408     }
7409     \seq_gput_right:Ne \g_@@_blocks_seq
7410     {
7411         \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.

```

7412     {
7413         \exp_not:n { #3 } ,
7414         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7415     \bool_if:NT \g_@@_rotate_bool
7416     {
7417         \bool_if:NTF \g_@@_rotate_c_bool
7418         { m }
7419         {
7420             \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7421             { T }

```

```

7422     }
7423   }
7424 }
7425 {
7426   \box_use_drop:c
7427   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7428 }
7429 }
7430 \bool_set_false:N \g_@@_rotate_c_bool
7431 }

7432 \cs_new:Npn \@@_adjust_hpos_rotate:
7433 {
7434   \bool_if:NT \g_@@_rotate_bool
7435   {
7436     \str_set:Nx \l_@@_hpos_block_str
7437     {
7438       \bool_if:NTF \g_@@_rotate_c_bool
7439       { c }
7440       {
7441         \str_case:onF \l_@@_vpos_block_str
7442         { b l B l t r T r }
7443         {
7444           \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
7445           { r }
7446           { l }
7447         }
7448       }
7449     }
7450   }
7451 }
7452 \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*.

```

7453 \cs_new_protected:Npn \@@_rotate_box_of_block:
7454 {
7455   \box_grotate:cn
7456   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7457   { 90 }
7458   \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7459   {
7460     \vbox_gset_top:cn
7461     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7462     {
7463       \skip_vertical:n { 0.8 ex }
7464       \box_use:c
7465       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7466     }
7467   }
7468   \bool_if:NT \g_@@_rotate_c_bool
7469   {
7470     \hbox_gset:cn
7471     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7472     {
7473       \c_math_toggle_token
7474       \vcenter
7475       {
7476         \box_use:c
7477         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7478       }
7479       \c_math_toggle_token

```



```

7480     }
7481   }
7482 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key `p`). In that case, the content of the block is *not* composed right now in a box. The composition in a box will be done further, just after the construction of the array (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn`).

#1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7483 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7484 {
7485   \seq_gput_right:Ne \g_@@_blocks_seq
7486   {
7487     \l_tmpa_tl
7488     { \exp_not:n { #3 } }
7489     {
7490       \bool_if:NTF \l_@@_tabular_bool
7491       {
7492         \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7493       \@@_reset_arraystretch:
7494       \exp_not:n
7495       {
7496         \dim_zero:N \extrarowheight
7497         #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).

```

7498       \bool_if:NT \c_@@_testphase_table_bool
7499       { \tag_stop:n { table } }
7500       \use:e
7501       {
7502         \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7503         { @ { } \l_@@_hpos_block_str @ { } }
7504       }
7505       #5
7506       \end { tabular }
7507     }
7508   \group_end:
7509 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7510   {
7511     \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7512       \@@_reset_arraystretch:
7513       \exp_not:n
7514       {
7515         \dim_zero:N \extrarowheight
7516         #4
7517         \c_math_toggle_token
7518         \use:e
7519         {
7520           \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7521           { @ { } \l_@@_hpos_block_str @ { } }
7522         }
7523         #5

```

```

7524         \end { array }
7525         \c_math_toggle_token
7526     }
7527     \group_end:
7528 }
7529 }
7530 }
7531 }
7532 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7533 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7534 {
7535     \seq_gput_right:Ne \g_@@_blocks_seq
7536     {
7537         \l_tmpa_tl
7538         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7539         { { \exp_not:n { #4 #5 } } }
7540     }
7541 }
7542 \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`.

```

7543 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7544 {
7545     \seq_gput_right:Ne \g_@@_blocks_seq
7546     {
7547         \l_tmpa_tl
7548         { \exp_not:n { #3 } }
7549         { \exp_not:n { #4 #5 } }
7550     }
7551 }
7552 \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).

```

7553 \keys_define:nn { nicematrix / Block / SecondPass }
7554 {
7555     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7556     ampersand-in-blocks .default:n = true ,
7557     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7558     tikz .code:n =
7559         \IfPackageLoadedTF { tikz }
7560         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7561         { \@@_error:n { tikz~key~without~tikz } } ,
7562     tikz .value_required:n = true ,
7563     fill .code:n =
7564         \tl_set_rescan:Nnn
7565         \l_@@_fill_tl
7566         { \char_set_catcode_other:N ! }
7567         { #1 } ,
7568     fill .value_required:n = true ,
7569     opacity .tl_set:N = \l_@@_opacity_tl ,
7570     opacity .value_required:n = true ,
7571     draw .code:n =
7572         \tl_set_rescan:Nnn

```

```

7573     \l_@@_draw_tl
7574     { \char_set_catcode_other:N ! }
7575     { #1 } ,
7576     draw .default:n = default ,
7577     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7578     rounded-corners .default:n = 4 pt ,
7579     color .code:n =
7580     \@@_color:n { #1 }
7581     \tl_set_rescan:Nnn
7582     \l_@@_draw_tl
7583     { \char_set_catcode_other:N ! }
7584     { #1 } ,
7585     borders .clist_set:N = \l_@@_borders_clist ,
7586     borders .value_required:n = true ,
7587     hvlines .meta:n = { vlines , hlines } ,
7588     vlines .bool_set:N = \l_@@_vlines_block_bool ,
7589     vlines .default:n = true ,
7590     hlines .bool_set:N = \l_@@_hlines_block_bool ,
7591     hlines .default:n = true ,
7592     line-width .dim_set:N = \l_@@_line_width_dim ,
7593     line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7594     j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7595     \bool_set_true:N \l_@@_p_block_bool ,
7596     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7597     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7598     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7599     L .code:n = \str_set:Nn \l_@@_hpos_block_str L
7600     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7601     R .code:n = \str_set:Nn \l_@@_hpos_block_str R
7602     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7603     C .code:n = \str_set:Nn \l_@@_hpos_block_str C
7604     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7605     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7606     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7607     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7608     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7609     m .code:n = \str_set:Nn \l_@@_vpos_block_str m ,
7610     m .value_forbidden:n = true ,
7611     v-center .meta:n = m ,
7612     p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7613     p .value_forbidden:n = true ,
7614     name .tl_set:N = \l_@@_block_name_str ,
7615     name .value_required:n = true ,
7616     name .initial:n = ,
7617     respect-arraystretch .code:n =
7618     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7619     respect-arraystretch .value_forbidden:n = true ,
7620     transparent .bool_set:N = \l_@@_transparent_bool ,
7621     transparent .default:n = true ,
7622     transparent .initial:n = false ,
7623     unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7624 }

```

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.

```

7625 \cs_new_protected:Npn \@@_draw_blocks:
7626 {
7627   \bool_if:NTF \c_@@_recent_array_bool
7628   { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7629   { \cs_set_eq:NN \ialign \@@_old_ialign: }

```

```

7630 \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7631 }
7632 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7633 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7634 \int_zero:N \l_@@_last_row_int
7635 \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”).

```

7636 \int_compare:nNnTF { #3 } > { 98 }
7637 { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7638 { \int_set:Nn \l_@@_last_row_int { #3 } }
7639 \int_compare:nNnTF { #4 } > { 98 }
7640 { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7641 { \int_set:Nn \l_@@_last_col_int { #4 } }
7642 \int_compare:nNnTF { \l_@@_last_col_int } > { \g_@@_col_total_int }
7643 {
7644   \bool_lazy_and:nnTF
7645     { \l_@@_preamble_bool }
7646     {
7647       \int_compare_p:n
7648         { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7649     }
7650   {
7651     \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7652     \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7653     \@@_msg_redirect_name:nn { columns-not-used } { none }
7654   }
7655   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7656 }
7657 {
7658   \int_compare:nNnTF { \l_@@_last_row_int } > { \g_@@_row_total_int }
7659   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7660   {
7661     \@@_Block_v:nneenn
7662       { #1 }
7663       { #2 }
7664       { \int_use:N \l_@@_last_row_int }
7665       { \int_use:N \l_@@_last_col_int }
7666       { #5 }
7667       { #6 }
7668   }
7669 }
7670 }

```

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

```

7671 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7672 {

```

The group is for the keys.

```

7673 \group_begin:
7674 \int_compare:nNnT { #1 } = { #3 }
7675 { \str_set:Nn \l_@@_vpos_block_str { t } }
7676 \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`.

```

7677 \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7678 \bool_lazy_and:nnT
7679 { \l_@@_vlines_block_bool }
7680 { ! \l_@@_ampersand_bool }
7681 {
7682   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7683   {
7684     \@@_vlines_block:nnn
7685     { \exp_not:n { #5 } }
7686     { #1 - #2 }
7687     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7688   }
7689 }
7690 \bool_if:NT \l_@@_hlines_block_bool
7691 {
7692   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7693   {
7694     \@@_hlines_block:nnn
7695     { \exp_not:n { #5 } }
7696     { #1 - #2 }
7697     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7698   }
7699 }
7700 \bool_if:NF \l_@@_transparent_bool
7701 {
7702   \bool_lazy_and:nnF { \l_@@_vlines_block_bool } { \l_@@_hlines_block_bool }
7703   {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

7704   \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7705   { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7706   }
7707 }
7708 \tl_if_empty:NF \l_@@_draw_tl
7709 {
7710   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7711   { \@@_error:n { hlines-with-color } }
7712   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7713   {
7714     \@@_stroke_block:nnn
7715     { \exp_not:n { #5 } }
7716     { #1 - #2 }
7717     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7718   }
7719   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7720   { { #1 } { #2 } { #3 } { #4 } }
7721 }
7722 \clist_if_empty:NF \l_@@_borders_clist
7723 {
7724   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7725   {
7726     \@@_stroke_borders_block:nnn
7727     { \exp_not:n { #5 } }
7728     { #1 - #2 }
7729     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7730   }
7731 }

```

#5 are the options

```

7732 \tl_if_empty:NF \l_@@_fill_tl
7733 {
7734   \@@_add_opacity_to_fill:
7735   \tl_gput_right:Ne \g_@@_pre_code_before_tl
7736   {
7737     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7738     { #1 - #2 }
7739     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7740     { \dim_use:N \l_@@_rounded_corners_dim }
7741   }
7742 }
7743 \seq_if_empty:NF \l_@@_tikz_seq
7744 {
7745   \tl_gput_right:Ne \g_nicematrix_code_before_tl
7746   {
7747     \@@_block_tikz:nnnnn
7748     { \seq_use:Nn \l_@@_tikz_seq { , } }
7749     { #1 }
7750     { #2 }
7751     { \int_use:N \l_@@_last_row_int }
7752     { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7753   }
7754 }
7755 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7756 {
7757   \tl_gput_right:Ne \g_@@_pre_code_after_tl
7758   {
7759     \@@_actually_diagbox:nnnnnn
7760     { #1 }
7761     { #2 }
7762     { \int_use:N \l_@@_last_row_int }
7763     { \int_use:N \l_@@_last_col_int }
7764     { \exp_not:n { ##1 } }
7765     { \exp_not:n { ##2 } }
7766   }
7767 }

```

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
three	four	two
six	seven	five
		eight

We highlight the node `1-1-block-short`

our block		one
three	four	two
six	seven	five
		eight

The construction of the node corresponding to the merged cells.

```

7768 \pgfpicture
7769 \pgfrememberpicturepositiononpagetrue

```

```

7770 \pgf@relevantforpicturesizefalse
7771 \@@_qpoint:n { row - #1 }
7772 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7773 \@@_qpoint:n { col - #2 }
7774 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7775 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7776 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7777 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7778 \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.

```

7779 \@@_pgf_rect_node:nnnnn
7780 { \@@_env: - #1 - #2 - block }
7781 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7782 \str_if_empty:NF \l_@@_block_name_str
7783 {
7784   \pgfnodealias
7785   { \@@_env: - \l_@@_block_name_str }
7786   { \@@_env: - #1 - #2 - block }
7787   \str_if_empty:NF \l_@@_name_str
7788   {
7789     \pgfnodealias
7790     { \l_@@_name_str - \l_@@_block_name_str }
7791     { \@@_env: - #1 - #2 - block }
7792   }
7793 }

```

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.

```

7794 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7795 {
7796   \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.

```

7797 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7798 {

```

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.

```

7799 \cs_if_exist:cT
7800 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7801 {
7802   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7803   {
7804     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7805     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7806   }
7807 }
7808 }

```

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.

```

7809 \dim_compare:nNnT { \l_tmpb_dim } = { \c_max_dim }
7810 {
7811   \@@_qpoint:n { col - #2 }
7812   \dim_set_eq:NN \l_tmpb_dim \pgf@x
7813 }
7814 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7815 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }

```

```

7816 {
7817   \cs_if_exist:cT
7818   { pgf @ sh @ ns @ \@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7819   {
7820     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7821     {
7822       \pgfpointanchor
7823       { \@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7824       { east }
7825       \dim_set:Nn \l_@@_tmpd_dim
7826       { \dim_max:nn { \l_@@_tmpd_dim } { \pgf@x } }
7827     }
7828   }
7829 }
7830 \dim_compare:nNnT { \l_@@_tmpd_dim } = { - \c_max_dim }
7831 {
7832   \@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7833   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7834 }
7835 \@_pgf_rect_node:nnnnn
7836 { \@_env: - #1 - #2 - block - short }
7837 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7838 }

```

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.

```

7839 \bool_if:NT \l_@@_medium_nodes_bool
7840 {
7841   \@_pgf_rect_node:nnn
7842   { \@_env: - #1 - #2 - block - medium }
7843   { \pgfpointanchor { \@_env: - #1 - #2 - medium } { north-west } }
7844   {
7845     \pgfpointanchor
7846     { \@_env:
7847       - \int_use:N \l_@@_last_row_int
7848       - \int_use:N \l_@@_last_col_int - medium
7849     }
7850     { south-east }
7851   }
7852 }
7853 \endpgfpicture

```

```

7854 \bool_if:NTF \l_@@_ampersand_bool
7855 {
7856   \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7857   \int_zero_new:N \l_@@_split_int
7858   \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7859   \pgfpicture
7860   \pgfrememberpicturepositiononpagetrue
7861   \pgf@relevantforpicturesizefalse
7862
7863   \@_qpoint:n { row - #1 }
7864   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7865   \@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7866   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7867   \@_qpoint:n { col - #2 }
7868   \dim_set_eq:NN \l_tmpa_dim \pgf@x
7869   \@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7870   \dim_set:Nn \l_tmpb_dim
7871   { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7872   \bool_lazy_or:nnT
7873   { \l_@@_vlines_block_bool }
7874   { \str_if_eq_p:ee \l_@@_vlines_clist { all } }

```



```

7875 {
7876   \int_step_inline:nn { \l_@@_split_int - 1 }
7877   {
7878     \pgfpathmoveto
7879     {
7880       \pgfpoint
7881       { \l_tmpa_dim + ##1 \l_tmpb_dim }
7882       \l_@@_tmpc_dim
7883     }
7884     \pgfpathlineto
7885     {
7886       \pgfpoint
7887       { \l_tmpa_dim + ##1 \l_tmpb_dim }
7888       \l_@@_tmpd_dim
7889     }
7890     \CT@arc@
7891     \pgfsetlinewidth { 1.1 \arrayrulewidth }
7892     \pgfsetrectcap
7893     \pgfusepathqstroke
7894   }
7895 }
7896 \@@_qpoint:n { row - #1 - base }
7897 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7898 \int_step_inline:nn { \l_@@_split_int }
7899 {
7900   \group_begin:
7901   \dim_set:Nn \col@sep
7902   { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
7903   \pgftransformshift
7904   {
7905     \pgfpoint
7906     {
7907       \l_tmpa_dim + ##1 \l_tmpb_dim -
7908       \str_case:on \l_@@_hpos_block_str
7909       {
7910         l { \l_tmpb_dim + \col@sep }
7911         c { 0.5 \l_tmpb_dim }
7912         r { \col@sep }
7913       }
7914     }
7915     { \l_@@_tmpc_dim }
7916   }
7917   \pgfset { inner~sep = \c_zero_dim }
7918   \pgfnode
7919   { rectangle }
7920   {
7921     \str_case:on \l_@@_hpos_block_str
7922     {
7923       c { base }
7924       l { base-west }
7925       r { base-east }
7926     }
7927   }
7928   { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { } { }
7929   \group_end:
7930 }
7931 \endpgfpicture
7932 }

```

Now the case where there is no ampersand & in the content of the block.

```

7933 {
7934   \bool_if:NTF \l_@@_p_block_bool
7935   {

```

When the final user has used the key `p`, we have to compute the width.

```

7936 \pgfpicture
7937 \pgfrememberpicturepositiononpagetrue
7938 \pgf@relevantforpicturesizefalse
7939 \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7940 {
7941   \@@_qpoint:n { col - #2 }
7942   \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7943   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7944 }
7945 {
7946   \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7947   \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7948   \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7949 }
7950 \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7951 \endpgfpicture
7952 \hbox_set:Nn \l_@@_cell_box
7953 {
7954   \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
7955   { \g_tmpb_dim }
7956   \str_case:on \l_@@_hpos_block_str
7957   { c \centering r \raggedleft l \raggedright j { } }
7958   #6
7959   \end { minipage }
7960 }
7961 }
7962 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7963 \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.

```

7964 \pgfpicture
7965 \pgfrememberpicturepositiononpagetrue
7966 \pgf@relevantforpicturesizefalse
7967 \bool_lazy_any:nTF
7968 {
7969   { \str_if_empty_p:N \l_@@_vpos_block_str }
7970   { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7971   { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7972   { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7973 }
7974 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7975 \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`.

```

7976 \bool_if:nT \g_@@_last_col_found_bool
7977 {
7978   \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
7979   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7980 }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7981 \tl_set:Ne \l_tmpa_tl
7982 {
7983   \str_case:on \l_@@_vpos_block_str
7984   {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7985         { } {
7986             \str_case:on \l_@@_hpos_block_str
7987             {
7988                 c { center }
7989                 l { west }
7990                 r { east }
7991                 j { center }
7992             }
7993         }
7994     c {
7995         \str_case:on \l_@@_hpos_block_str
7996         {
7997             c { center }
7998             l { west }
7999             r { east }
8000             j { center }
8001         }
8002     }
8003
8004     T {
8005         \str_case:on \l_@@_hpos_block_str
8006         {
8007             c { north }
8008             l { north-west }
8009             r { north-east }
8010             j { north }
8011         }
8012     }
8013
8014     B {
8015         \str_case:on \l_@@_hpos_block_str
8016         {
8017             c { south }
8018             l { south-west }
8019             r { south-east }
8020             j { south }
8021         }
8022     }
8023
8024     }
8025
8026     \pgftransformshift
8027     {
8028         \pgfpointanchor
8029         {
8030             \@@_env: - #1 - #2 - block
8031             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8032         }
8033         { \l_tmpa_tl }
8034     }
8035     \pgfset { inner~sep = \c_zero_dim }
8036     \pgfnode
8037     { rectangle }
8038     { \l_tmpa_tl }
8039     { \box_use_drop:N \l_@@_cell_box } { } { }
8040 }

```

End of the case when `\l_@@_vpos_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```

8041     {
8042         \pgfextracty \l_tmpa_dim
8043         {

```

```

8044         \@@_qpoint:n
8045         {
8046             row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
8047             - base
8048         }
8049     }
8050     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in \pgf@x) the x -value of the center of the block.

```

8051     \pgfpointanchor
8052     {
8053         \@@_env: - #1 - #2 - block
8054         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8055     }
8056     {
8057         \str_case:on \l_@@_hpos_block_str
8058         {
8059             c { center }
8060             l { west }
8061             r { east }
8062             j { center }
8063         }
8064     }

```

We put the label of the block which has been composed in \l_@@_cell_box.

```

8065     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8066     \pgfset { inner~sep = \c_zero_dim }
8067     \pgfnode
8068     { rectangle }
8069     {
8070         \str_case:on \l_@@_hpos_block_str
8071         {
8072             c { base }
8073             l { base~west }
8074             r { base~east }
8075             j { base }
8076         }
8077     }
8078     { \box_use_drop:N \l_@@_cell_box } { } { }
8079 }
8080 \endpgfpicture
8081 }
8082 \group_end:
8083 }
8084 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }

```

For the command \cellcolor used within a sub-cell of a \Block (when the character & is used inside the cell).

```

8085 \cs_set_protected:Npn \@@_fill:nnnnn #1 #2 #3 #4 #5
8086 {
8087     \pgfpicture
8088     \pgfrememberpicturepositiononpagetrue
8089     \pgf@relevantforpicturesizefalse
8090     \pgfpathrectanglecorners
8091     { \pgfpoint { #2 } { #3 } }
8092     { \pgfpoint { #4 } { #5 } }
8093     \pgfsetfillcolor { #1 }
8094     \pgfusepath { fill }
8095     \endpgfpicture
8096 }

```

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).

```

8097 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8098 {
8099   \tl_if_empty:NF \l_@@_opacity_tl
8100   {
8101     \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8102       {
8103         \tl_set:Ne \l_@@_fill_tl
8104         {
8105           [ opacity = \l_@@_opacity_tl ,
8106             \tl_tail:o \l_@@_fill_tl
8107         }
8108       }
8109       {
8110         \tl_set:Ne \l_@@_fill_tl
8111         { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8112       }
8113     }
8114   }

```

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).

```

8115 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8116 {
8117   \group_begin:
8118   \tl_clear:N \l_@@_draw_tl
8119   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8120   \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8121   \pgfpicture
8122   \pgfrememberpicturepositiononpagetrue
8123   \pgf@relevantforpicturesizefalse
8124   \tl_if_empty:NF \l_@@_draw_tl
8125   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8126     \tl_if_eq:NnTF \l_@@_draw_tl { default }
8127     { \CT@arc@ }
8128     { \@@_color:o \l_@@_draw_tl }
8129   }
8130   \pgfsetcornersarced
8131   {
8132     \pgfpoint
8133     { \l_@@_rounded_corners_dim }
8134     { \l_@@_rounded_corners_dim }
8135   }
8136   \@@_cut_on_hyphen:w #2 \q_stop
8137   \int_compare:nNnF { \l_tmpa_tl } > { \c@iRow }
8138   {
8139     \int_compare:nNnF { \l_tmpb_tl } > { \c@jCol }
8140     {
8141       \@@_qpoint:n { row - \l_tmpa_tl }
8142       \dim_set_eq:NN \l_tmpb_dim \pgf@y
8143       \@@_qpoint:n { col - \l_tmpb_tl }
8144       \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8145       \@@_cut_on_hyphen:w #3 \q_stop
8146       \int_compare:nNnT { \l_tmpa_tl } > { \c@iRow }
8147       { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8148       \int_compare:nNnT { \l_tmpb_tl } > { \c@jCol }
8149       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }

```

```

8150 \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8151 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8152 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8153 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8154 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8155 \pgfpathrectanglecorners
8156 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8157 { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8158 \dim_compare:nNnTF { \l_@@_rounded_corners_dim } = { \c_zero_dim }
8159 { \pgfusepathqstroke }
8160 { \pgfusepath { stroke } }
8161 }
8162 }
8163 \endpgfpicture
8164 \group_end:
8165 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8166 \keys_define:nn { nicematrix / BlockStroke }
8167 {
8168   color .tl_set:N = \l_@@_draw_tl ,
8169   draw .code:n =
8170     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8171   draw .default:n = default ,
8172   line-width .dim_set:N = \l_@@_line_width_dim ,
8173   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8174   rounded-corners .default:n = 4 pt
8175 }

```

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).

```

8176 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8177 {
8178   \group_begin:
8179   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8180   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8181   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8182   \@@_cut_on_hyphen:w #2 \q_stop
8183   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8184   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8185   \@@_cut_on_hyphen:w #3 \q_stop
8186   \tl_set:Nc \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8187   \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8188   \int_step_inline:nnn { \l_@@_tmpd_tl } { \l_tmpb_tl }
8189   {
8190     \use:e
8191     {
8192       \@@_vline:n
8193       {
8194         position = ##1 ,
8195         start = \l_@@_tmpc_tl ,
8196         end = \int_eval:n { \l_tmpa_tl - 1 } ,
8197         total-width = \dim_use:N \l_@@_line_width_dim
8198       }
8199     }
8200   }
8201   \group_end:
8202 }
8203 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8204 {
8205   \group_begin:
8206   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth

```

```

8207 \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8208 \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8209 \@@_cut_on_hyphen:w #2 \q_stop
8210 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8211 \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8212 \@@_cut_on_hyphen:w #3 \q_stop
8213 \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8214 \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8215 \int_step_inline:nnn { \l_@@_tmpc_tl } { \l_tmpa_tl }
8216 {
8217   \use:e
8218   {
8219     \@@_hline:n
8220     {
8221       position = ##1 ,
8222       start = \l_@@_tmpd_tl ,
8223       end = \int_eval:n { \l_tmpb_tl - 1 } ,
8224       total-width = \dim_use:N \l_@@_line_width_dim
8225     }
8226   }
8227 }
8228 \group_end:
8229 }

```

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).

```

8230 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8231 {
8232   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8233   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8234   \dim_compare:nNnTF { \l_@@_rounded_corners_dim } > { \c_zero_dim }
8235   { \@@_error:n { borders~forbidden } }
8236   {
8237     \tl_clear_new:N \l_@@_borders_tikz_tl
8238     \keys_set:no
8239     { nicematrix / OnlyForTikzInBorders }
8240     \l_@@_borders_clist
8241     \@@_cut_on_hyphen:w #2 \q_stop
8242     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8243     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8244     \@@_cut_on_hyphen:w #3 \q_stop
8245     \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8246     \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8247     \@@_stroke_borders_block_i:
8248   }
8249 }
8250 \hook_gput_code:nnn { begindocument } { . }
8251 {
8252   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8253   {
8254     \c_@@_pgfortikzpicture_tl
8255     \@@_stroke_borders_block_ii:
8256     \c_@@_endpgfortikzpicture_tl
8257   }
8258 }
8259 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8260 {
8261   \pgfrememberpicturerepositiononpagetrue
8262   \pgf@relevantforpicturesizefalse
8263   \CT@arc@
8264   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

```

8265 \clist_if_in:NnT \l_@@_borders_clist { right }
8266 { \@@_stroke_vertical:n \l_tmpb_tl }
8267 \clist_if_in:NnT \l_@@_borders_clist { left }
8268 { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8269 \clist_if_in:NnT \l_@@_borders_clist { bottom }
8270 { \@@_stroke_horizontal:n \l_tmpa_tl }
8271 \clist_if_in:NnT \l_@@_borders_clist { top }
8272 { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8273 }
8274 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8275 {
8276   tikz .code:n =
8277     \cs_if_exist:NTF \tikzpicture
8278     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8279     { \@@_error:n { tikz~in~borders~without~tikz } } ,
8280   tikz .value_required:n = true ,
8281   top .code:n = ,
8282   bottom .code:n = ,
8283   left .code:n = ,
8284   right .code:n = ,
8285   unknown .code:n = \@@_error:n { bad~border }
8286 }

```

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).

```

8287 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8288 {
8289   \@@_qpoint:n \l_@@_tmpc_tl
8290   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8291   \@@_qpoint:n \l_tmpa_tl
8292   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8293   \@@_qpoint:n { #1 }
8294   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8295   {
8296     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8297     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8298     \pgfusepathqstroke
8299   }
8300   {
8301     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8302     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8303   }
8304 }

```

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).

```

8305 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8306 {
8307   \@@_qpoint:n \l_@@_tmpd_tl
8308   \clist_if_in:NnTF \l_@@_borders_clist { left }
8309   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8310   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8311   \@@_qpoint:n \l_tmpb_tl
8312   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8313   \@@_qpoint:n { #1 }
8314   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8315   {
8316     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8317     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8318     \pgfusepathqstroke
8319   }
8320   {
8321     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }

```



```

8322      ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8323    }
8324  }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

8325 \keys_define:nn { nicematrix / BlockBorders }
8326 {
8327   borders .clist_set:N = \l_@@_borders_clist ,
8328   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8329   rounded-corners .default:n = 4 pt ,
8330   line-width .dim_set:N = \l_@@_line_width_dim
8331 }

```

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.

```

8332 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8333 {
8334   \begin { tikzpicture }
8335   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because **#5** is a list of lists.

```

8336   \clist_map_inline:nn { #1 }
8337   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8338     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8339     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8340     (
8341       [
8342         xshift = \dim_use:N \l_@@_offset_dim ,
8343         yshift = - \dim_use:N \l_@@_offset_dim
8344       ]
8345       #2 -| #3
8346     )
8347     rectangle
8348     (
8349       [
8350         xshift = - \dim_use:N \l_@@_offset_dim ,
8351         yshift = \dim_use:N \l_@@_offset_dim
8352       ]
8353       \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8354     ) ;
8355   }
8356 \end { tikzpicture }
8357 }
8358 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

8359 \keys_define:nn { nicematrix / SpecialOffset }
8360 { 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.

```

8361 \cs_new_protected:Npn \@@_NullBlock:
8362 { \@@_collect_options:n { \@@_NullBlock_i: } }
8363 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8364 { }

```

27 How to draw the dotted lines transparently

```

8365 \cs_set_protected:Npn \@@_renew_matrix:
8366 {
8367   \RenewDocumentEnvironment { pmatrix } { }
8368     { \pNiceMatrix }
8369     { \endpNiceMatrix }
8370   \RenewDocumentEnvironment { vmatrix } { }
8371     { \vNiceMatrix }
8372     { \endvNiceMatrix }
8373   \RenewDocumentEnvironment { Vmatrix } { }
8374     { \VNiceMatrix }
8375     { \endVNiceMatrix }
8376   \RenewDocumentEnvironment { bmatrix } { }
8377     { \bNiceMatrix }
8378     { \endbNiceMatrix }
8379   \RenewDocumentEnvironment { Bmatrix } { }
8380     { \BNiceMatrix }
8381     { \endBNiceMatrix }
8382 }

```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8383 \keys_define:nn { nicematrix / Auto }
8384 {
8385   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8386   columns-type .value_required:n = true ,
8387   l .meta:n = { columns-type = l } ,
8388   r .meta:n = { columns-type = r } ,
8389   c .meta:n = { columns-type = c } ,
8390   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8391   delimiters / color .value_required:n = true ,
8392   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8393   delimiters / max-width .default:n = true ,
8394   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8395   delimiters .value_required:n = true ,
8396   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8397   rounded-corners .default:n = 4 pt
8398 }
8399 \NewDocumentCommand \AutoNiceMatrixWithDelims
8400 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8401 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
8402 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8403 {

```

The group is for the protection of the keys.

```

8404   \group_begin:
8405   \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8406   \use:e
8407   {
8408     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8409       { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8410       [ \exp_not:o \l_tmpa_tl ]
8411   }
8412   \int_if_zero:nT { \l_@@_first_row_int }
8413   {
8414     \int_if_zero:nT { \l_@@_first_col_int } { & }
8415     \prg_replicate:nn { #4 - 1 } { & }
8416     \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\\

```

```

8417     }
8418     \prg_replicate:nn { #3 }
8419     {
8420         \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).

```

8421         \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8422         \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8423     }
8424     \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
8425     {
8426         \int_if_zero:nT { \l_@@_first_col_int } { & }
8427         \prg_replicate:nn { #4 - 1 } { & }
8428         \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8429     }
8430     \end { NiceArrayWithDelims }
8431     \group_end:
8432 }
8433 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
8434 {
8435     \cs_set_protected:cpn { #1 AutoNiceMatrix }
8436     {
8437         \bool_gset_true:N \g_@@_delims_bool
8438         \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8439         \AutoNiceMatrixWithDelims { #2 } { #3 }
8440     }
8441 }

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8442 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
8443 {
8444     \group_begin:
8445     \bool_gset_false:N \g_@@_delims_bool
8446     \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8447     \group_end:
8448 }

```

29 The redefinition of the command \dotfill

```

8449 \cs_set_eq:NN \@@_old_dotfill: \dotfill
8450 \cs_new_protected:Npn \@@_dotfill:
8451 {

```

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}).

```

8452     \@@_old_dotfill:
8453     \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8454 }

```

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.

```

8455 \cs_new_protected:Npn \@@_dotfill_i:
8456 {
8457     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = { \c_zero_dim }
8458     { \@@_old_dotfill: }
8459 }

```

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.

```

8460 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8461 {
8462   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8463   {
8464     \@@_actually_diagbox:nnnnnn
8465     { \int_use:N \c@iRow }
8466     { \int_use:N \c@jCol }
8467     { \int_use:N \c@iRow }
8468     { \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.

```

8469     { \g_@@_row_style_tl \exp_not:n { #1 } }
8470     { \g_@@_row_style_tl \exp_not:n { #2 } }
8471   }

```

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`.

```

8472   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8473   {
8474     { \int_use:N \c@iRow }
8475     { \int_use:N \c@jCol }
8476     { \int_use:N \c@iRow }
8477     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8478     { }
8479   }
8480 }

```

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.

```

8481 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8482 {
8483   \pgfpicture
8484   \pgf@relevantforpicturesizefalse
8485   \pgfrememberpicturepositiononpagetrue
8486   \@@_qpoint:n { row - #1 }
8487   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8488   \@@_qpoint:n { col - #2 }
8489   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8490   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8491   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8492   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8493   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8494   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8495   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8496   {

```

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.

```

8497   \CT@arc@
8498   \pgfsetroundcap

```

```

8499     \pgfusepathqstroke
8500   }
8501   \pgfset { inner~sep = 1 pt }
8502   \pgfscope
8503   \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8504   \pgfnode { rectangle } { south~west }
8505   {
8506     \begin { minipage } { 20 cm }

```

The `\scan_stop`: avoids an error in math mode when the argument #5 is empty.

```

8507     \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8508   \end { minipage }
8509   }
8510   { }
8511   { }
8512 \endpgfscope
8513 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8514 \pgfnode { rectangle } { north~east }
8515 {
8516   \begin { minipage } { 20 cm }
8517   \raggedleft
8518   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8519   \end { minipage }
8520 }
8521 { }
8522 { }
8523 \endpgfpicture
8524 }

```

31 The keyword `\CodeAfter`

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment `{@@-light-syntax}` on p. 85.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

8525 \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 `\`.

```

8526 \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`.

```

8527 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8528 {
8529   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8530   \@@_CodeAfter_iv:n
8531 }

```

We catch the argument of the command `\end` (in #1).

```

8532 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8533 {

```

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.

```

8534   \str_if_eq:eeTF \@currentenv { #1 }
8535   { \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`.

```

8536     {
8537         \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8538         \@@_CodeAfter_ii:n
8539     }
8540 }
```

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.

```

8541 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8542 {
8543     \pgfpicture
8544     \pgfrememberpicturepositiononpagetrue
8545     \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.

```

8546     \@@_qpoint:n { row - 1 }
8547     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8548     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8549     \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).

```

8550     \bool_if:nTF { #3 }
8551     { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8552     { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8553     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8554     {
8555         \cs_if_exist:cT
8556         { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8557         {
8558             \pgfpointanchor
8559             { \@@_env: - ##1 - #2 }
8560             { \bool_if:nTF { #3 } { west } { east } }
8561             \dim_set:Nn \l_tmpa_dim
8562             {
8563                 \bool_if:nTF { #3 }
8564                 { \dim_min:nn }
8565                 { \dim_max:nn }
8566                 \l_tmpa_dim
8567                 { \pgf@x }
8568             }
8569         }
8570     }
```

Now we can put the delimiter with a node of PGF.

```

8571 \pgfset { inner~sep = \c_zero_dim }
8572 \dim_zero:N \nulldelimiterspace
8573 \pgftransformshift
8574 {
8575   \pgfpoint
8576   { \l_tmpa_dim }
8577   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8578 }
8579 \pgfnode
8580 { rectangle }
8581 { \bool_if:nTF { #3 } { east } { west } }
8582 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8583 \nullfont
8584 \c_math_toggle_token
8585 \@@_color:o \l_@@_delimiters_color_tl
8586 \bool_if:nTF { #3 } { \left #1 } { \left . }
8587 \vcenter
8588 {
8589   \nullfont
8590   \hrule \@height
8591   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8592   \@depth \c_zero_dim
8593   \@width \c_zero_dim
8594 }
8595 \bool_if:nTF { #3 } { \right . } { \right #1 }
8596 \c_math_toggle_token
8597 }
8598 { }
8599 { }
8600 \endpgfpicture
8601 }

```

33 The command \SubMatrix

```

8602 \keys_define:nn { nicematrix / sub-matrix }
8603 {
8604   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8605   extra-height .value_required:n = true ,
8606   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8607   left-xshift .value_required:n = true ,
8608   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8609   right-xshift .value_required:n = true ,
8610   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8611   xshift .value_required:n = true ,
8612   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8613   delimiters / color .value_required:n = true ,
8614   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8615   slim .default:n = true ,
8616   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8617   hlines .default:n = all ,
8618   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8619   vlines .default:n = all ,
8620   hvlines .meta:n = { hlines, vlines } ,
8621   hvlines .value_forbidden:n = true
8622 }
8623 \keys_define:nn { nicematrix }
8624 {
8625   SubMatrix .inherit:n = nicematrix / sub-matrix ,

```

```

8626 NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8627 pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8628 NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8629 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8630 \keys_define:nn { nicematrix / SubMatrix }
8631 {
8632   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8633   delimiters / color .value_required:n = true ,
8634   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8635   hlines .default:n = all ,
8636   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8637   vlines .default:n = all ,
8638   hvlines .meta:n = { hlines, vlines } ,
8639   hvlines .value_forbidden:n = true ,
8640   name .code:n =
8641     \tl_if_empty:nTF { #1 }
8642     { \@@_error:n { Invalid-name } }
8643     {
8644       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8645       {
8646         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8647         { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
8648         {
8649           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8650           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8651         }
8652       }
8653       { \@@_error:n { Invalid-name } }
8654     } ,
8655   name .value_required:n = true ,
8656   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8657   rules .value_required:n = true ,
8658   code .tl_set:N = \l_@@_code_tl ,
8659   code .value_required:n = true ,
8660   unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8661 }

8662 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8663 {
8664   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8665   {
8666     \SubMatrix { #1 } { #2 } { #3 } { #4 }
8667     [
8668       delimiters / color = \l_@@_delimiters_color_tl ,
8669       hlines = \l_@@_submatrix_hlines_clist ,
8670       vlines = \l_@@_submatrix_vlines_clist ,
8671       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8672       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8673       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8674       slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8675       #5
8676     ]
8677   }
8678   \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8679   \ignorespaces
8680 }

8681 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8682 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8683 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

```



```

8684 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8685 {
8686   \seq_gput_right:Ne \g_@@_submatrix_seq
8687   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8688     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8689     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8690     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8691     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8692   }
8693 }

```

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).

```

8694 \NewDocumentCommand \@@_compute_i_j:nn
8695 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8696 { \@@_compute_i_j:nnnn #1 #2 }

```

```

8697 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8698 {
8699   \def \l_@@_first_i_tl { #1 }
8700   \def \l_@@_first_j_tl { #2 }
8701   \def \l_@@_last_i_tl { #3 }
8702   \def \l_@@_last_j_tl { #4 }
8703   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8704     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8705   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8706     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8707   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8708     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8709   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8710     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8711 }

```

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`.

```

8712 \hook_gput_code:nnn { begindocument } { . }
8713 {
8714   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
8715   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
8716     { \@@_sub_matrix:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
8717 }
8718 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
8719 {
8720   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8721 \@@_compute_i_j:nn { #2 } { #3 }
8722 \int_compare:nNt { \l_@@_first_i_tl } = { \l_@@_last_i_tl }
8723 { \def \arraystretch { 1 } }
8724 \bool_lazy_or:nnTF
8725 { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
8726 { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
8727 { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8728 {
8729 \str_clear_new:N \l_@@_submatrix_name_str
8730 \keys_set:nn { nicematrix / SubMatrix } { #5 }
8731 \pgfpicture
8732 \pgfrememberpicturepositiononpagetrue
8733 \pgf@relevantforpicturesizefalse
8734 \pgfset { inner~sep = \c_zero_dim }
8735 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8736 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currying.

```

8737 \bool_if:Ntf \l_@@_submatrix_slim_bool
8738 { \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl } }
8739 { \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int } }
8740 {
8741 \cs_if_exist:cT
8742 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8743 {
8744 \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8745 \dim_compare:nNt { \pgf@x } < { \l_@@_x_initial_dim }
8746 { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8747 }
8748 \cs_if_exist:cT
8749 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8750 {
8751 \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8752 \dim_compare:nNt { \pgf@x } > { \l_@@_x_final_dim }
8753 { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8754 }
8755 }
8756 \dim_compare:nNtTF { \l_@@_x_initial_dim } = { \c_max_dim }
8757 { \@@_error:nn { Impossible~delimiter } { left } }
8758 {
8759 \dim_compare:nNtTF { \l_@@_x_final_dim } = { - \c_max_dim }
8760 { \@@_error:nn { Impossible~delimiter } { right } }
8761 { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8762 }
8763 \endpgfpicture
8764 }
8765 \group_end:
8766 \ignorespaces
8767 }

```

`#1` is the left delimiter, `#2` is the right one, `#3` is the subscript and `#4` is the superscript.

```

8768 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8769 {
8770 \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8771 \dim_set:Nn \l_@@_y_initial_dim
8772 {
8773 \fp_to_dim:n
8774 {
8775 \pgf@y
8776 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8777 }
8778 }

```

```

8779 \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8780 \dim_set:Nn \l_@@_y_final_dim
8781 { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8782 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
8783 {
8784   \cs_if_exist:cT
8785   { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8786   {
8787     \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8788     \dim_set:Nn \l_@@_y_initial_dim
8789     { \dim_max:nn { \l_@@_y_initial_dim } { \pgf@y } }
8790   }
8791   \cs_if_exist:cT
8792   { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8793   {
8794     \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8795     \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
8796     { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8797   }
8798 }
8799 \dim_set:Nn \l_tmpa_dim
8800 {
8801   \l_@@_y_initial_dim - \l_@@_y_final_dim +
8802   \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8803 }
8804 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8805 \group_begin:
8806 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8807 \@@_set_CTarc:o \l_@@_rules_color_tl
8808 \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`.

```

8809 \seq_map_inline:Nn \g_@@_cols_vlism_seq
8810 {
8811   \int_compare:nNnT { \l_@@_first_j_tl } < { ##1 }
8812   {
8813     \int_compare:nNnT
8814     { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8815     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8816 \@@_qpoint:n { col - ##1 }
8817 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8818 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8819 \pgfusepathqstroke
8820 }
8821 }
8822 }

```

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.

```

8823 \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
8824 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8825 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8826 {
8827   \bool_lazy_and:nnTF
8828   { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
8829   {
8830     \int_compare_p:nNn

```

```

8831         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8832     {
8833         \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8834         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8835         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8836         \pgfusepathqstroke
8837     }
8838     { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8839 }

```

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.

```

8840 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
8841 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8842 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8843 {
8844     \bool_lazy_and:nnTF
8845     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
8846     {
8847         \int_compare_p:nNn
8848         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8849     {
8850         \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8851     \group_begin:
We compute in \l_tmpa_dim the x-value of the left end of the rule.
8852     \dim_set:Nn \l_tmpa_dim
8853     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8854     \str_case:nn { #1 }
8855     {
8856         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8857         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8858         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8859     }
8860     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the *x*-value of the right end of the rule.

```

8861     \dim_set:Nn \l_tmpb_dim
8862     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8863     \str_case:nn { #2 }
8864     {
8865         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8866         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8867         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8868     }
8869     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8870     \pgfusepathqstroke
8871     \group_end:
8872 }
8873 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8874 }

```

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).

```

8875 \str_if_empty:NF \l_@@_submatrix_name_str
8876 {
8877     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8878     \l_@@_x_initial_dim \l_@@_y_initial_dim
8879     \l_@@_x_final_dim \l_@@_y_final_dim
8880 }
8881 \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`.

```

8882   \begin { pgfscope }
8883   \pgftransformshift
8884   {
8885     \pgfpoint
8886     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8887     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8888   }
8889   \str_if_empty:NTF \l_@@_submatrix_name_str
8890   { \@@_node_left:nn #1 { } }
8891   { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8892   \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8893   \pgftransformshift
8894   {
8895     \pgfpoint
8896     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8897     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8898   }
8899   \str_if_empty:NTF \l_@@_submatrix_name_str
8900   { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8901   {
8902     \@@_node_right:nnnn #2
8903     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8904   }

```

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`.

```

8905   \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8906   \flag_clear_new:N \l_@@_code_flag
8907   \l_@@_code_tl
8908 }

```

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`.

```

8909 \cs_set_eq:NN \@@_old_pgfpntanchor: \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by currying.

```

8910 \cs_new:Npn \@@_pgfpointanchor:n #1
8911 { \exp_args:Ne \@@_old_pgfpntanchor: { \@@_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`).

```

8912 \cs_new:Npn \@@_pgfpointanchor_i:n #1
8913 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }

```

```

8914 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
8915 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

8916 \str_if_empty:nTF { #1 }

```

First, when the name of the name begins with `\tikz@pp@name`.

```

8917 { \@@_pgfpointanchor_iv:w #2 }

```

And now, when there is no `\tikz@pp@name`.

```

8918 { \@@_pgfpointanchor_ii:n { #1 } }
8919 }

```

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`).

```

8920 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
8921 { \@@_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`.

```

8922 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

```

```

8923 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
8924 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

8925 \str_if_empty:nTF { #2 }

```

First the case where the argument does *not* contain an hyphen.

```

8926 { \@@_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`).

```

8927 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8928 }

```

The following function is for the case when the name contains an hyphen.

```

8929 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8930 {

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

8931 \@@_env:
8932 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8933 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8934 }

```

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.

```

8935 \tl_const:Nn \c_@@_integers_alist_tl
8936 {
8937 { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8938 { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8939 { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8940 { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8941 }

```

```

8942 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
8943 {

```

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`.

```

8944 \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8945 {
8946   \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

8947 \@@_env: -
8948 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8949 { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8950 { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8951 }
8952 {
8953   \str_if_eq:eeTF { #1 } { last }
8954   {
8955     \flag_raise:N \l_@@_code_flag
8956     \@@_env: -
8957     \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8958     { \int_eval:n { \l_@@_last_i_tl + 1 } }
8959     { \int_eval:n { \l_@@_last_j_tl + 1 } }
8960   }
8961   { #1 }
8962 }
8963 }

```

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`).

```

8964 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8965 {
8966   \pgfnode
8967   { rectangle }
8968   { east }
8969   {
8970     \nullfont
8971     \c_math_toggle_token
8972     \@@_color:o \l_@@_delimiters_color_tl
8973     \left #1
8974     \vcenter
8975     {
8976       \nullfont
8977       \hrule \@height \l_tmpa_dim
8978       \c_zero_dim
8979       \width \c_zero_dim
8980     }
8981     \right .
8982     \c_math_toggle_token
8983   }
8984   { #2 }
8985   { }
8986 }

```

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.

```

8987 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4

```

```

8988 {
8989   \pgfnode
8990   { rectangle }
8991   { west }
8992   {
8993     \nullfont
8994     \c_math_toggle_token
8995     \colorlet { current-color } { . }
8996     \@@_color:o \l_@@_delimiters_color_tl
8997     \left .
8998     \vcenter
8999     {
9000       \nullfont
9001       \hrule \@height \l_tmpa_dim
9002               \@depth \c_zero_dim
9003               \@width \c_zero_dim
9004     }
9005     \right #1
9006     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9007     ^ { \color { current-color } \smash { #4 } }
9008     \c_math_toggle_token
9009   }
9010   { #2 }
9011   { }
9012 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9013 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m O { } }
9014 {
9015   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9016   \ignorespaces
9017 }
9018 \NewDocumentCommand \@@_OverBrace { 0 { } m m m O { } }
9019 {
9020   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9021   \ignorespaces
9022 }
9023 \keys_define:nn { nicematrix / Brace }
9024 {
9025   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9026   left-shorten .default:n = true ,
9027   left-shorten .value_forbidden:n = true ,
9028   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9029   right-shorten .default:n = true ,
9030   right-shorten .value_forbidden:n = true ,
9031   shorten .meta:n = { left-shorten , right-shorten } ,
9032   shorten .value_forbidden:n = true ,
9033   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9034   yshift .value_required:n = true ,
9035   yshift .initial:n = \c_zero_dim ,
9036   color .tl_set:N = \l_tmpa_tl ,
9037   color .value_required:n = true ,
9038   unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
9039 }

```


#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`.

```

9040 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9041 {
9042   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9043   \@@_compute_i_j:nn { #1 } { #2 }
9044   \bool_lazy_or:nnTF
9045     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9046     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9047     {
9048       \str_if_eq:eeTF { #5 } { under }
9049       { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9050       { \@@_error:nn { Construct-too-large } { \OverBrace } }
9051     }
9052   {
9053     \tl_clear:N \l_tmpa_tl
9054     \keys_set:nn { nicematrix / Brace } { #4 }
9055     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9056     \pgfpicture
9057     \pgfrememberpicturepositiononpagetrue
9058     \pgf@relevantforpicturesizefalse
9059     \bool_if:NT \l_@@_brace_left_shorten_bool
9060     {
9061       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9062       \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9063       {
9064         \cs_if_exist:cT
9065           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9066           {
9067             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9068
9069             \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9070             { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9071           }
9072       }
9073     }
9074     \bool_lazy_or:nnT
9075       { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9076       { \dim_compare_p:nNn { \l_@@_x_initial_dim } = { \c_max_dim } }
9077     {
9078       \@@_qpoint:n { col - \l_@@_first_j_tl }
9079       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9080     }
9081     \bool_if:NT \l_@@_brace_right_shorten_bool
9082     {
9083       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9084       \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9085       {
9086         \cs_if_exist:cT
9087           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9088           {
9089             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9090             \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
9091             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9092           }
9093       }
9094     }
9095     \bool_lazy_or:nnT
9096       { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9097       { \dim_compare_p:nNn { \l_@@_x_final_dim } = { - \c_max_dim } }
9098     {

```

```

9099         \l_@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9100         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9101     }
9102     \pgfset { inner~sep = \c_zero_dim }
9103     \str_if_eq:eeTF { #5 } { under }
9104         { \l_@@_underbrace_i:n { #3 } }
9105         { \l_@@_overbrace_i:n { #3 } }
9106     \endpgfpicture
9107 }
9108 \group_end:
9109 }

```

The argument is the text to put above the brace.

```

9110 \cs_new_protected:Npn \l_@@_overbrace_i:n #1
9111 {
9112     \l_@@_qpoint:n { row - \l_@@_first_i_tl }
9113     \pgftransformshift
9114     {
9115         \pgfpoint
9116         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9117         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9118     }
9119     \pgfnode
9120     { rectangle }
9121     { south }
9122     {
9123         \vtop
9124         {
9125             \group_begin:
9126             \everycr { }
9127             \halign
9128             {
9129                 \hfil ## \hfil \crcr
9130                 \bool_if:NTF \l_@@_tabular_bool
9131                     { \begin { tabular } { c } #1 \end { tabular } }
9132                     { $ \begin { array } { c } #1 \end { array } $ }
9133                 \cr
9134                 \c_math_toggle_token
9135                 \overbrace
9136                 {
9137                     \hbox_to_wd:nn
9138                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9139                     { }
9140                 }
9141                 \c_math_toggle_token
9142                 \cr
9143             }
9144             \group_end:
9145         }
9146     }
9147     { }
9148     { }
9149 }

```

The argument is the text to put under the brace.

```

9150 \cs_new_protected:Npn \l_@@_underbrace_i:n #1
9151 {
9152     \l_@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9153     \pgftransformshift
9154     {
9155         \pgfpoint
9156         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9157         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }

```

```

9158     }
9159   \pgfnode
9160   { rectangle }
9161   { north }
9162   {
9163     \group_begin:
9164     \everycr { }
9165     \vbox
9166     {
9167       \halign
9168       {
9169         \hfil ## \hfil \crcr
9170         \c_math_toggle_token
9171         \underbrace
9172         {
9173           \hbox_to_wd:nn
9174           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9175           { }
9176         }
9177         \c_math_toggle_token
9178         \cr
9179         \bool_if:NTF \l_@@_tabular_bool
9180         { \begin { tabular } { c } #1 \end { tabular } }
9181         { $ \begin { array } { c } #1 \end { array } $ }
9182         \cr
9183       }
9184     }
9185     \group_end:
9186   }
9187   { }
9188   { }
9189 }

```

35 The commands HBrace et VBrace

```

9190 \hook_gput_code:nnn { begindocument } { . . }
9191 {
9192   \cs_if_exist:cT { tikz@library@decorations.pathreplacing@loaded }
9193   {
9194     \tikzset
9195     {
9196       nicematrix / brace / .style =
9197       {
9198         decoration = { brace , raise = -0.15 em } ,
9199         decorate ,
9200       } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9201       nicematrix / mirrored-brace / .style =
9202       {
9203         nicematrix / brace ,
9204         decoration = mirror ,
9205       }
9206     }
9207   }
9208 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9209 \keys_define:nn { nicematrix / Hbrace }
9210 {
9211   color .code:n = ,
9212   horizontal-label .code:n = ,
9213   horizontal-labels .code:n = ,
9214   shorten .code:n = ,
9215   shorten-start .code:n = ,
9216   shorten-end .code:n = ,
9217   unknown .code:n = \@_error:n { Unknown~key~for~Hbrace }
9218 }

```

Here we need an “fully expandable” command.

```

9219 \NewExpandableDocumentCommand { \@_Hbrace } { 0 { } m m }
9220 {
9221   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9222   { \@_hbrace:nnn { #1 } { #2 } { #3 } }
9223   { \@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9224 }

```

The following command must *not* be protected.

```

9225 \cs_new:Npn \@_hbrace:nnn #1 #2 #3
9226 {
9227   \int_compare:nNnTF { \c@iRow } < { \c_one_int }
9228   {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9229   \str_if_eq:nnTF { #2 } { * }
9230   {
9231     \NiceMatrixOptions { nullify-dots }
9232     \Ldots
9233     [
9234       line-style = nicematrix / brace ,
9235       #1 ,
9236       up =
9237       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9238     ]
9239   }
9240   {
9241     \Hdotsfor
9242     [
9243       line-style = nicematrix / brace ,
9244       #1 ,
9245       up =
9246       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9247     ]
9248     { #2 }
9249   }
9250 }
9251 {
9252   \str_if_eq:nnTF { #2 } { * }
9253   {
9254     \NiceMatrixOptions { nullify-dots }
9255     \Ldots
9256     [
9257       line-style = nicematrix / mirrored-brace ,
9258       #1 ,
9259       down =
9260       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9261     ]
9262   }
9263   {
9264     \Hdotsfor
9265     [
9266       line-style = nicematrix / mirrored-brace ,

```

```

9267         #1 ,
9268         down =
9269         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9270     ]
9271     { #2 }
9272 }
9273 }
9274 \keys_set:nn { nicematrix / Hbrace } { #1 }
9275 }

```

Here we need an “fully expandable” command.

```

9276 \NewExpandableDocumentCommand { \@@_Vbrace } { 0 { } m m }
9277 {
9278   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9279   { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9280   { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9281 }

```

The following command must *not* be protected.

```

9282 \cs_new:Npn \@@_vbrace:nnn #1 #2 #3
9283 {
9284   \int_if_zero:nTF { \c@jCol }
9285   {
9286     \str_if_eq:nnTF { #2 } { * }
9287     {
9288       \NiceMatrixOptions { nullify-dots }
9289       \Vdots
9290       [
9291         Vbrace ,
9292         line-style = nicematrix / mirrored-brace ,
9293         #1 ,
9294         down =
9295         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9296       ]
9297     }
9298     {
9299       \Vdotsfor
9300       [
9301         Vbrace ,
9302         line-style = nicematrix / mirrored-brace ,
9303         #1 ,
9304         down =
9305         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9306       ]
9307     { #2 }
9308   }
9309 }
9310 {
9311   \str_if_eq:nnTF { #2 } { * }
9312   {
9313     \NiceMatrixOptions { nullify-dots }
9314     \Vdots
9315     [
9316       Vbrace ,
9317       line-style = nicematrix / brace ,
9318       #1 ,
9319       up =
9320       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9321     ]
9322   }
9323   {
9324     \Vdotsfor
9325     [
9326       Vbrace ,

```

```

9327         line-style = nicematrix / brace ,
9328         #1 ,
9329         up =
9330         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9331     ]
9332     { #2 }
9333 }
9334 }
9335 \keys_set:nn { nicematrix / Hbrace } { #1 }
9336 }

```

36 The command TikzEveryCell

```

9337 \bool_new:N \l_@@_not_empty_bool
9338 \bool_new:N \l_@@_empty_bool
9339
9340 \keys_define:nn { nicematrix / TikzEveryCell }
9341 {
9342     not-empty .code:n =
9343         \bool_lazy_or:nnTF
9344         { \l_@@_in_code_after_bool }
9345         { \g_@@_create_cell_nodes_bool }
9346         { \bool_set_true:N \l_@@_not_empty_bool }
9347         { \@@_error:n { detection-of-empty-cells } } ,
9348     not-empty .value_forbidden:n = true ,
9349     empty .code:n =
9350         \bool_lazy_or:nnTF
9351         { \l_@@_in_code_after_bool }
9352         { \g_@@_create_cell_nodes_bool }
9353         { \bool_set_true:N \l_@@_empty_bool }
9354         { \@@_error:n { detection-of-empty-cells } } ,
9355     empty .value_forbidden:n = true ,
9356     unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
9357 }
9358
9359
9360 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9361 {
9362     \IfPackageLoadedTF { tikz }
9363     {
9364         \group_begin:
9365         \keys_set:nn { nicematrix / TikzEveryCell } { #1 }

```

The inner pair of braces in the following line is mandatory because, the last argument of `\@@_tikz:nnnnn` is a *list of lists* of TikZ keys.

```

9366         \tl_set:Nn \l_tmpa_tl { { #2 } }
9367         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9368         { \@@_for_a_block:nnnnn ##1 }
9369         \@@_all_the_cells:
9370         \group_end:
9371     }
9372     { \@@_error:n { TikzEveryCell~without~tikz } }
9373 }
9374
9375 \tl_new:N \l_@@_i_tl
9376 \tl_new:N \l_@@_j_tl
9377
9378
9379 \cs_new_protected:Nn \@@_all_the_cells:
9380 {
9381     \int_step_inline:nn \c@iRow
9382     {

```

```

9383 \int_step_inline:nn \c@jCol
9384 {
9385   \cs_if_exist:cF { cell - ##1 - #####1 }
9386   {
9387     \clist_if_in:NcF \l_@@_corners_cells_clist
9388     { ##1 - #####1 }
9389     {
9390       \bool_set_false:N \l_tmpa_bool
9391       \cs_if_exist:cTF
9392       { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
9393       {
9394         \bool_if:NF \l_@@_empty_bool
9395         { \bool_set_true:N \l_tmpa_bool }
9396       }
9397       {
9398         \bool_if:NF \l_@@_not_empty_bool
9399         { \bool_set_true:N \l_tmpa_bool }
9400       }
9401       \bool_if:NT \l_tmpa_bool
9402       {
9403         \@@_block_tikz:nnnnn
9404         \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
9405       }
9406     }
9407   }
9408 }
9409 }
9410 }
9411
9412 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9413 {
9414   \bool_if:NF \l_@@_empty_bool
9415   {
9416     \@@_block_tikz:nnnnn
9417     \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9418   }
9419   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9420 }
9421
9422 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9423 {
9424   \int_step_inline:nnn { #1 } { #3 }
9425   {
9426     \int_step_inline:nnn { #2 } { #4 }
9427     { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9428   }
9429 }

```

37 The command \ShowCellNames

```

9430 \NewDocumentCommand \@@_ShowCellNames { }
9431 {
9432   \bool_if:NT \l_@@_in_code_after_bool
9433   {
9434     \pgfpicture
9435     \pgfrememberpicturerepositiononpagetrue
9436     \pgf@relevantforpicturesizefalse
9437     \pgfpathrectanglecorners
9438     { \@@_qpoint:n { 1 } }
9439     {
9440       \@@_qpoint:n
9441       { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }

```

```

9442     }
9443     \pgfsetfillopacity { 0.75 }
9444     \pgfsetfillcolor { white }
9445     \pgfusepathqfill
9446     \endpgfpicture
9447 }
9448 \dim_gzero_new:N \g_@@_tmpc_dim
9449 \dim_gzero_new:N \g_@@_tmpd_dim
9450 \dim_gzero_new:N \g_@@_tmpe_dim
9451 \int_step_inline:nn { \c@iRow }
9452 {
9453     \bool_if:NTF \l_@@_in_code_after_bool
9454     {
9455         \pgfpicture
9456         \pgfrememberpicturepositiononpagetrue
9457         \pgf@relevantforpicturesizefalse
9458     }
9459     { \begin { pgfpicture } }
9460     \@@_qpoint:n { row - ##1 }
9461     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9462     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9463     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9464     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9465     \bool_if:NTF \l_@@_in_code_after_bool
9466     { \endpgfpicture }
9467     { \end { pgfpicture } }
9468     \int_step_inline:nn { \c@jCol }
9469     {
9470         \hbox_set:Nn \l_tmpa_box
9471         {
9472             \normalfont \Large \sffamily \bfseries
9473             \bool_if:NTF \l_@@_in_code_after_bool
9474             { \color { red } }
9475             { \color { red ! 50 } }
9476             ##1 - ####1
9477         }
9478         \bool_if:NTF \l_@@_in_code_after_bool
9479         {
9480             \pgfpicture
9481             \pgfrememberpicturepositiononpagetrue
9482             \pgf@relevantforpicturesizefalse
9483         }
9484         { \begin { pgfpicture } }
9485         \@@_qpoint:n { col - ####1 }
9486         \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9487         \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9488         \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9489         \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9490         \bool_if:NTF \l_@@_in_code_after_bool
9491         { \endpgfpicture }
9492         { \end { pgfpicture } }
9493         \fp_set:Nn \l_tmpa_fp
9494         {
9495             \fp_min:nn
9496             {
9497                 \fp_min:nn
9498                 { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9499                 { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9500             }
9501             { 1.0 }
9502         }
9503         \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9504         \pgfpicture

```



```

9505     \pgfrememberpicturepositiononpagetrue
9506     \pgf@relevantforpicturesizefalse
9507     \pgftransformshift
9508     {
9509         \pgfpoint
9510         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9511         { \dim_use:N \g_tmpa_dim }
9512     }
9513     \pgfnode
9514     { rectangle }
9515     { center }
9516     { \box_use:N \l_tmpa_box }
9517     { }
9518     { }
9519     \endpgfpicture
9520 }
9521 }
9522 }

```

38 We process the options at package loading

We process the options when the package is loaded (with `\usepackage`) but we recommend to use `\NiceMatrixOptions` instead.

We must process these options after the definition of the environment `{NiceMatrix}` because the option `renew-matrix` executes the code `\cs_set_eq:NN \env@matrix \NiceMatrix`.

Of course, the command `\NiceMatrix` must be defined before such an instruction is executed.

The boolean `\g_@@_footnotehyper_bool` will indicate if the option `footnotehyper` is used.

```

9523 \bool_new:N \g_@@_footnotehyper_bool

```

The boolean `\g_@@_footnote_bool` will indicate if the option `footnote` is used, but quickly, it will also be set to true if the option `footnotehyper` is used.

```

9524 \bool_new:N \g_@@_footnote_bool

9525 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9526 {
9527     You-have-used-the-key~' \l_keys_key_str '~when-loading-nicematrix~
9528     but-that-key-is-unknown. \\
9529     It-will-be-ignored. \\
9530     For-a-list-of-the-available-keys,~type-H~<return>.
9531 }
9532 {
9533     The-available-keys-are~(in~alphabetic~order):~
9534     footnote,~
9535     footnotehyper,~
9536     messages-for-Overleaf,~
9537     renew-dots~and~
9538     renew-matrix.
9539 }

9540 \keys_define:nn { nicematrix }
9541 {
9542     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9543     renew-dots .value_forbidden:n = true ,
9544     renew-matrix .code:n = \@@_renew_matrix: ,
9545     renew-matrix .value_forbidden:n = true ,
9546     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9547     footnote .bool_set:N = \g_@@_footnote_bool ,
9548     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9549     unknown .code:n = \@@_error:n { Unknown~key~for~package }
9550 }
9551 \ProcessKeyOptions

```

```

9552 \@@_msg_new:nn { footnote-with-footnotehyper-package }
9553 {
9554   You~can't~use~the~option~'footnote'~because~the~package~
9555   footnotehyper~has~already~been~loaded.~
9556   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9557   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9558   of~the~package~footnotehyper.\\
9559   The~package~footnote~won't~be~loaded.
9560 }
9561 \@@_msg_new:nn { footnotehyper-with-footnote-package }
9562 {
9563   You~can't~use~the~option~'footnotehyper'~because~the~package~
9564   footnote~has~already~been~loaded.~
9565   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9566   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9567   of~the~package~footnote.\\
9568   The~package~footnotehyper~won't~be~loaded.
9569 }
9570 \bool_if:NT \g_@@_footnote_bool
9571 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9572   \IfClassLoadedTF { beamer }
9573   { \bool_set_false:N \g_@@_footnote_bool }
9574   {
9575     \IfPackageLoadedTF { footnotehyper }
9576     { \@@_error:n { footnote-with-footnotehyper-package } }
9577     { \usepackage { footnote } }
9578   }
9579 }
9580 \bool_if:NT \g_@@_footnotehyper_bool
9581 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9582   \IfClassLoadedTF { beamer }
9583   { \bool_set_false:N \g_@@_footnote_bool }
9584   {
9585     \IfPackageLoadedTF { footnote }
9586     { \@@_error:n { footnotehyper-with-footnote-package } }
9587     { \usepackage { footnotehyper } }
9588   }
9589   \bool_set_true:N \g_@@_footnote_bool
9590 }

```

The flag `\g_@@_footnote_bool` is raised and so, we will only have to test `\g_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

39 About the package `underscore`

If the user loads the package `underscore`, it must be loaded *before* the package `nicematrix`. If it is loaded after, we raise an error.

```

9591 \bool_new:N \l_@@_underscore_loaded_bool
9592 \IfPackageLoadedT { underscore }
9593 { \bool_set_true:N \l_@@_underscore_loaded_bool }

```

```

9594 \hook_gput_code:nnn { begindocument } { . }
9595 {
9596   \bool_if:NF \l_@@_underscore_loaded_bool
9597   {
9598     \IfPackageLoadedT { underscore }
9599     { \@@_error:n { underscore~after~nicematrix } }
9600   }
9601 }

```

40 Error messages of the package

```

9602 \str_const:Ne \c_@@_available_keys_str
9603 {
9604   \bool_if:nTF { ! \g_@@_messages_for_Overleaf_bool }
9605   { For~a~list~of~the~available~keys,~type~H~<return>. }
9606   { }
9607 }
9608 \seq_new:N \g_@@_types_of_matrix_seq
9609 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9610 {
9611   NiceMatrix ,
9612   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9613 }
9614 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9615 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_error_too_much_cols:` is triggered. This command raises an error but also tries to give the best information to the user in the error message. The command `\seq_if_in:NoF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

9616 \cs_new_protected:Npn \@@_error_too_much_cols:
9617 {
9618   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9619   { \@@_fatal:nn { too~much~cols~for~array } }
9620   \int_compare:nNnT { \l_@@_last_col_int } = { -2 }
9621   { \@@_fatal:n { too~much~cols~for~matrix } }
9622   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
9623   { \@@_fatal:n { too~much~cols~for~matrix } }
9624   \bool_if:NF \l_@@_last_col_without_value_bool
9625   { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
9626 }

```

The following command must *not* be protected since it's used in an error message.

```

9627 \cs_new:Npn \@@_message_hdotsfor:
9628 {
9629   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9630   { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ is~incorrect. }
9631 }
9632 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9633 {
9634   Incompatible~options.\\
9635   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~the~same~time.\\
9636   The~output~will~not~be~reliable.
9637 }
9638 \@@_msg_new:nn { key~color~inside }
9639 {
9640   Key~deprecated.\\
9641   The~key~'color~inside'~(and~its~alias~'colortbl~like')~is~now~point~less~

```

```

9642     and-have-been-deprecated.\\
9643     You-won't-have-similar-message-till-the-end-of-the-document.
9644 }

9645 \@@_msg_new:nn { invalid-weight }
9646 {
9647     Unknown-key.\\
9648     The-key~' \l_keys_key_str '-of-your-column-X-is-unknown-and-will-be-ignored.
9649 }

9650 \@@_msg_new:nn { last-col-not-used }
9651 {
9652     Column-not-used.\\
9653     The-key~'last-col'-is-in-force-but-you-have-not-used-that-last-column~
9654     in~your~\@@_full_name_env: .~
9655     However,~you-can~go~on.
9656 }

9657 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9658 {
9659     Too-much-columns.\\
9660     In~the~row~ \int_eval:n { \c@iRow },~
9661     you~try~to~use~more~columns~
9662     than~allowed~by~your~ \@@_full_name_env: .
9663     \@@_message_hdotsfor: \
9664     The~maximal~number~of~columns~is~ \int_eval:n { \l_@@_last_col_int - 1 }~
9665     (plus~the~exterior~columns).~This~error~is~fatal.
9666 }

9667 \@@_msg_new:nn { too-much-cols-for-matrix }
9668 {
9669     Too-much-columns.\\
9670     In~the~row~ \int_eval:n { \c@iRow },~
9671     you~try~to~use~more~columns~than~allowed~by~your~ \@@_full_name_env: .
9672     \@@_message_hdotsfor: \
9673     Recall~that~the~maximal~number~of~columns~for~a~matrix~
9674     (excepted~the~potential~exterior~columns)~is~fixed~by~the~
9675     LaTeX~counter~'MaxMatrixCols'.~
9676     Its~current~value~is~ \int_use:N \c@MaxMatrixCols \
9677     (use~ \token_to_str:N \setcounter \ to~change~that~value).~
9678     This~error~is~fatal.
9679 }

9680 \@@_msg_new:nn { too-much-cols-for-array }
9681 {
9682     Too-much-columns.\\
9683     In~the~row~ \int_eval:n { \c@iRow },~
9684     ~you~try~to~use~more~columns~than~allowed~by~your~
9685     \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal~number~of~columns~is~
9686     \int_use:N \g_@@_static_num_of_col_int \
9687     \bool_if:nT
9688     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
9689     { ~(plus~the~exterior~ones) }
9690     since~the~preamble~is~' \g_@@_user_preamble_tl '.\\
9691     This~error~is~fatal.
9692 }

9693 \@@_msg_new:nn { columns-not-used }
9694 {
9695     Columns-not-used.\\
9696     The~preamble~of~your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '.~
9697     It~announces~ \int_use:N \g_@@_static_num_of_col_int \
9698     columns~but~you~only~used~ \int_use:N \c@jCol .\\
9699     The~columns~you~did~not~used~won't~be~created.\\
9700     You-won't-have-similar-warning-till-the-end-of-the-document.
9701 }

```

```

9702 \@@_msg_new:nn { empty-preamble }
9703 {
9704   Empty-preamble.\
9705   The-preamble-of-your~ \@@_full_name_env: \ is-empty.\
9706   This-error-is-fatal.
9707 }
9708 \@@_msg_new:nn { in-first-col }
9709 {
9710   Erroneous-use.\
9711   You-can't-use-the-command~#1 in-the-first-column~(number~0)~of~the-array.\
9712   That-command-will-be-ignored.
9713 }
9714 \@@_msg_new:nn { in-last-col }
9715 {
9716   Erroneous-use.\
9717   You-can't-use-the-command~#1 in-the-last-column~(exterior)~of~the-array.\
9718   That-command-will-be-ignored.
9719 }
9720 \@@_msg_new:nn { in-first-row }
9721 {
9722   Erroneous-use.\
9723   You-can't-use-the-command~#1 in-the-first-row~(number~0)~of~the-array.\
9724   That-command-will-be-ignored.
9725 }
9726 \@@_msg_new:nn { in-last-row }
9727 {
9728   Erroneous-use.\
9729   You-can't-use-the-command~#1 in-the-last-row~(exterior)~of~the-array.\
9730   That-command-will-be-ignored.
9731 }
9732 \@@_msg_new:nn { TopRule~without~booktabs }
9733 {
9734   Erroneous-use.\
9735   You-can't-use-the-command~ #1 because~'booktabs'~is-not-loaded.\
9736   That-command-will-be-ignored.
9737 }
9738 \@@_msg_new:nn { TopRule~without~tikz }
9739 {
9740   Erroneous-use.\
9741   You-can't-use-the-command~ #1 because~'tikz'~is-not-loaded.\
9742   That-command-will-be-ignored.
9743 }
9744 \@@_msg_new:nn { caption~outside~float }
9745 {
9746   Key~caption~forbidden.\
9747   You-can't-use-the-key~'caption'~because~you~are~not~in~a~floating~
9748   environment~(such~as~\{table\})~.~This-key~will~be~ignored.
9749 }
9750 \@@_msg_new:nn { short-caption~without~caption }
9751 {
9752   You-should-not-use-the-key~'short-caption'~without~'caption'.~
9753   However,~your~'short-caption'~will~be~used~as~'caption'.
9754 }
9755 \@@_msg_new:nn { double-closing-delimiter }
9756 {
9757   Double~delimiter.\
9758   You-can't-put-a-second-closing-delimiter~"#1"~just~after~a~first~closing~
9759   delimiter.~This-delimiter~will~be~ignored.
9760 }

```

```

9761 \@@_msg_new:nn { delimiter~after~opening }
9762 {
9763   Double~delimiter.\\
9764   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9765   delimiter.~That~delimiter~will~be~ignored.
9766 }
9767 \@@_msg_new:nn { bad~option~for~line~style }
9768 {
9769   Bad~line~style.\\
9770   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9771   is~'standard'.~That~key~will~be~ignored.
9772 }
9773 \@@_msg_new:nn { corners~with~no~cell~nodes }
9774 {
9775   Incompatible~keys.\\
9776   You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
9777   is~in~force.\\
9778   If~you~go~on,~that~key~will~be~ignored.
9779 }
9780 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
9781 {
9782   Incompatible~keys.\\
9783   You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
9784   is~in~force.\\
9785   If~you~go~on,~those~extra~nodes~won't~be~created.
9786 }
9787 \@@_msg_new:nn { Identical~notes~in~caption }
9788 {
9789   Identical~tabular~notes.\\
9790   You~can't~put~several~notes~with~the~same~content~in~
9791   \token_to_str:N \caption \ (but~you~can~in~the~main~tabular).\\
9792   If~you~go~on,~the~output~will~probably~be~erroneous.
9793 }
9794 \@@_msg_new:nn { tabularnote~below~the~tabular }
9795 {
9796   \token_to_str:N \tabularnote \ forbidden\\
9797   You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
9798   of~your~tabular~because~the~caption~will~be~composed~below~
9799   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9800   key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .\\
9801   Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
9802   no~similar~error~will~raised~in~this~document.
9803 }
9804 \@@_msg_new:nn { Unknown~key~for~rules }
9805 {
9806   Unknown~key.\\
9807   There~is~only~two~keys~available~here:~width~and~color.\\
9808   Your~key~' \l_keys_key_str ' ~will~be~ignored.
9809 }
9810 \@@_msg_new:nn { Unknown~key~for~Hbrace }
9811 {
9812   Unknown~key.\\
9813   You~have~used~the~key~' \l_keys_key_str ' ~but~the~only~
9814   keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
9815   and~ \token_to_str:N \Vbrace \ are:~'color',~
9816   'horizontal~label(s)',~'shorten'~'shorten~end'~
9817   and~'shorten~start'.
9818 }
9819 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9820 {

```

```

9821     Unknown~key.\\
9822     There~is~only~two~keys~available~here::~
9823     'empty'~and~'not-empty'.\\
9824     Your~key~' \l_keys_key_str '~will~be~ignored.
9825 }
9826 \@@_msg_new:nn { Unknown~key~for~rotate }
9827 {
9828     Unknown~key.\\
9829     The~only~key~available~here~is~'c'.\\
9830     Your~key~' \l_keys_key_str '~will~be~ignored.
9831 }
9832 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9833 {
9834     Unknown~key.\\
9835     The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
9836     It~you~go~on,~you~will~probably~have~other~errors. \\
9837     \c_@@_available_keys_str
9838 }
9839 {
9840     The~available~keys~are~(in~alphabetic~order):~
9841     ccommand,~
9842     color,~
9843     command,~
9844     dotted,~
9845     letter,~
9846     multiplicity,~
9847     sep-color,~
9848     tikz,~and~total-width.
9849 }
9850 \@@_msg_new:nnn { Unknown~key~for~xdots }
9851 {
9852     Unknown~key.\\
9853     The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9854     \c_@@_available_keys_str
9855 }
9856 {
9857     The~available~keys~are~(in~alphabetic~order):~
9858     'color',~
9859     'horizontal(s)-labels',~
9860     'inter',~
9861     'line-style',~
9862     'radius',~
9863     'shorten',~
9864     'shorten-end'~and~'shorten-start'.
9865 }
9866 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9867 {
9868     Unknown~key.\\
9869     As~for~now,~there~is~only~two~keys~available~here::~'cols'~and~'respect-blocks'~
9870     (and~you~try~to~use~' \l_keys_key_str '~)\\
9871     That~key~will~be~ignored.
9872 }
9873 \@@_msg_new:nn { label~without~caption }
9874 {
9875     You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
9876     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9877 }
9878 \@@_msg_new:nn { W~warning }
9879 {
9880     Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
9881     (row~ \int_use:N \c@iRow ).

```

```

9882 }
9883 \@@_msg_new:nn { Construct-too-large }
9884 {
9885   Construct-too-large.\
9886   Your-command~ \token_to_str:N #1
9887   can't-be-drawn-because-your-matrix-is-too-small.\
9888   That-command-will-be-ignored.
9889 }
9890 \@@_msg_new:nn { underscore-after-nicematrix }
9891 {
9892   Problem-with-'underscore'.\
9893   The-package-'underscore'~should-be-loaded-before-'nicematrix'.~
9894   You-can-go-on-but-you-won't-be-able-to-write-something-such-as:\
9895   ' \token_to_str:N \Cdots \token_to_str:N _
9896   \{ n \token_to_str:N \text \{ ~times \} \}'
9897 }
9898 \@@_msg_new:nn { ampersand-in-light-syntax }
9899 {
9900   Ampersand-forbidden.\
9901   You-can't-use-an-ampersand~( \token_to_str:N & )~to-separate-columns-because~
9902   ~the-key-'light-syntax'~is-in-force.~This-error-is-fatal.
9903 }
9904 \@@_msg_new:nn { double-backslash-in-light-syntax }
9905 {
9906   Double-backslash-forbidden.\
9907   You-can't-use~ \token_to_str:N \
9908   ~to-separate-rows-because-the-key-'light-syntax'~
9909   is-in-force.~You-must-use-the-character~' \l_@@_end_of_row_tl '~
9910   (set-by-the-key-'end-of-row').~This-error-is-fatal.
9911 }
9912 \@@_msg_new:nn { hlines-with-color }
9913 {
9914   Incompatible-keys.\
9915   You-can't-use-the-keys-'hlines',~'vlines'~or~'hvlines'~for-a~
9916   \token_to_str:N \Block \ when-the-key-'color'~or~'draw'~is-used.\
9917   However,~you-can-put-several-commands~ \token_to_str:N \Block.\
9918   Your-key-will-be-discarded.
9919 }
9920 \@@_msg_new:nn { bad-value-for-baseline }
9921 {
9922   Bad-value-for-baseline.\
9923   The-value-given-to-'baseline'~( \int_use:N \l_tmpa_int )~is-not~
9924   valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int~and~
9925   \int_use:N \g_@@_row_total_int \ or-equal-to~'t',~'c'~or~'b'~or~of~
9926   the-form-'line-i'.\
9927   A-value-of-1-will-be-used.
9928 }
9929 \@@_msg_new:nn { detection-of-empty-cells }
9930 {
9931   Problem-with-'not-empty'~\
9932   For-technical-reasons,~you-must-activate~
9933   'create-cell-nodes'~in~ \token_to_str:N \CreateBefore \
9934   in-order-to-use-the-key~' \l_keys_key_str '~.\
9935   That-key-will-be-ignored.
9936 }
9937 \@@_msg_new:nn { siunitx-not-loaded }
9938 {
9939   siunitx-not-loaded~\
9940   You-can't-use-the-columns~'S'~because~'siunitx'~is-not-loaded.\
9941   That-error-is-fatal.

```



```

9942 }
9943 \@@_msg_new:nn { Invalid~name }
9944 {
9945   Invalid~name.\\
9946   You~can't~give~the~name~' \l_keys_value_tl '~to~a~ \token_to_str:N
9947   \SubMatrix \ of~your~ \@@_full_name_env: .\\
9948   A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9949   This~key~will~be~ignored.
9950 }
9951 \@@_msg_new:nn { Hbrace~not~allowed }
9952 {
9953   Command~not~allowed.\\
9954   You~can't~use~the~command~ \token_to_str:N #1
9955   because~you~have~not~loaded~
9956   \IfPackageLoadedTF { tikz }
9957   { the~TikZ~library~'decorations.pathreplacing'~Use~ }
9958   { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
9959   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\
9960   That~command~will~be~ignored.
9961 }
9962 \@@_msg_new:nn { Vbrace~not~allowed }
9963 {
9964   Command~not~allowed.\\
9965   You~can't~use~the~command~ \token_to_str:N \Vbrace \
9966   because~you~have~not~loaded~TikZ~
9967   and~the~TikZ~library~'decorations.pathreplacing'.\\
9968   Use: ~\token_to_str:N \usepackage \{tikz\}~
9969   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
9970   That~command~will~be~ignored.
9971 }
9972 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9973 {
9974   Wrong~line.\\
9975   You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9976   \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
9977   number~is~not~valid.~It~will~be~ignored.
9978 }
9979 \@@_msg_new:nn { Impossible~delimiter }
9980 {
9981   Impossible~delimiter.\\
9982   It's~impossible~to~draw~the~#1~delimiter~of~your~
9983   \token_to_str:N \SubMatrix \ because~all~the~cells~are~empty~
9984   in~that~column.
9985   \bool_if:NT \l_@@_submatrix_slim_bool
9986   { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9987   This~ \token_to_str:N \SubMatrix \ will~be~ignored.
9988 }
9989 \@@_msg_new:nnn { width~without~X~columns }
9990 {
9991   You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
9992   the~preamble~(' \g_@@_user_preamble_tl '~of~your~ \@@_full_name_env: .\\
9993   That~key~will~be~ignored.
9994 }
9995 {
9996   This~message~is~the~message~'width~without~X~columns'~
9997   of~the~module~'nicematrix'.~
9998   The~experimented~users~can~disable~that~message~with~
9999   \token_to_str:N \msg_redirect_name:nnn .\\
10000 }
10001
10002 \@@_msg_new:nn { key~multiplicity~with~dotted }

```

```

10003 {
10004     Incompatible~keys. \\
10005     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
10006     in~a~'custom~line'.~They~are~incompatible. \\
10007     The~key~'multiplicity'~will~be~discarded.
10008 }
10009 \@@_msg_new:nn { empty~environment }
10010 {
10011     Empty~environment.\\
10012     Your~ \@@_full_name_env: \ is~empty.~This~error~is~fatal.
10013 }
10014 \@@_msg_new:nn { No~letter~and~no~command }
10015 {
10016     Erroneous~use.\\
10017     Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
10018     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10019     '~'ccommand'~(to~draw~horizontal~rules).\\
10020     However,~you~can~go~on.
10021 }
10022 \@@_msg_new:nn { Forbidden~letter }
10023 {
10024     Forbidden~letter.\\
10025     You~can't~use~the~letter~'#1'~for~a~customized~line.~
10026     It~will~be~ignored.\\
10027     The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10028 }
10029 \@@_msg_new:nn { Several~letters }
10030 {
10031     Wrong~name.\\
10032     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10033     have~used~' \l_@@_letter_str ').\\
10034     It~will~be~ignored.
10035 }
10036 \@@_msg_new:nn { Delimiter~with~small }
10037 {
10038     Delimiter~forbidden.\\
10039     You~can't~put~a~delimiter~in~the~preamble~of~your~
10040     \@@_full_name_env: \
10041     because~the~key~'small'~is~in~force.\\
10042     This~error~is~fatal.
10043 }
10044 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10045 {
10046     Unknown~cell.\\
10047     Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10048     the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10049     can't~be~executed~because~a~cell~doesn't~exist.\\
10050     This~command~ \token_to_str:N \line \ will~be~ignored.
10051 }
10052 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10053 {
10054     Duplicate~name.\\
10055     The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10056     in~this~ \@@_full_name_env: .\\
10057     This~key~will~be~ignored.\\
10058     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10059     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10060 }
10061 {
10062     The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
10063     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .

```

```

10064 }
10065 \@@_msg_new:nn { r~or~l~with~preamble }
10066 {
10067   Erroneous~use.\\
10068   You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
10069   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10070   your~ \@@_full_name_env: .\\
10071   This~key~will~be~ignored.
10072 }
10073 \@@_msg_new:nn { Hdotsfor~in~col~0 }
10074 {
10075   Erroneous~use.\\
10076   You~can't~use~ \token_to_str:N \Hdotsfor \ in~an~exterior~column~of~
10077   the~array.~This~error~is~fatal.
10078 }
10079 \@@_msg_new:nn { bad~corner }
10080 {
10081   Bad~corner.\\
10082   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10083   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
10084   This~specification~of~corner~will~be~ignored.
10085 }
10086 \@@_msg_new:nn { bad~border }
10087 {
10088   Bad~border.\\
10089   \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
10090   (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
10091   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10092   also~use~the~key~'tikz'
10093   \IfPackageLoadedF { tikz }
10094   { ~if~you~load~the~LaTeX~package~'tikz' } ).\\
10095   This~specification~of~border~will~be~ignored.
10096 }
10097 \@@_msg_new:nn { TikzEveryCell~without~tikz }
10098 {
10099   TikZ~not~loaded.\\
10100   You~can't~use~ \token_to_str:N \TikzEveryCell \
10101   because~you~have~not~loaded~tikz.~
10102   This~command~will~be~ignored.
10103 }
10104 \@@_msg_new:nn { tikz~key~without~tikz }
10105 {
10106   TikZ~not~loaded.\\
10107   You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N
10108   \Block '~because~you~have~not~loaded~tikz.~
10109   This~key~will~be~ignored.
10110 }
10111 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
10112 {
10113   Erroneous~use.\\
10114   In~the~ \@@_full_name_env: ,~you~must~use~the~key~
10115   'last~col'~without~value.\\
10116   However,~you~can~go~on~for~this~time~
10117   (the~value~' \l_keys_value_tl '~will~be~ignored).
10118 }
10119 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
10120 {
10121   Erroneous~use. \\
10122   In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
10123   'last~col'~without~value. \\

```

```

10124     However,~you-can-go-on-for-this-time~
10125     (the-value~' \l_keys_value_t1 '~will-be-ignored).
10126 }

10127 \@@_msg_new:nn { Block-too-large-1 }
10128 {
10129     Block-too-large. \\
10130     You-try-to-draw-a-block-in-the-cell~#1-#2-of-your-matrix-but-the-matrix-is~
10131     too-small-for-that-block. \\
10132     This-block-and-maybe-others-will-be-ignored.
10133 }

10134 \@@_msg_new:nn { Block-too-large-2 }
10135 {
10136     Block-too-large. \\
10137     The-preamble-of-your~ \@@_full_name_env: \ announces~ \int_use:N
10138     \g_@@_static_num_of_col_int \
10139     columns-but-you-use-only~ \int_use:N \c_jCol \ and-that's-why-a-block~
10140     specified-in-the-cell~#1-#2-can't-be-drawn.~You-should-add-some-ampersands~
10141     (&)~at-the-end-of-the-first-row-of-your~ \@@_full_name_env: . \\
10142     This-block-and-maybe-others-will-be-ignored.
10143 }

10144 \@@_msg_new:nn { unknown-column-type }
10145 {
10146     Bad-column-type. \\
10147     The-column-type~'#1'~in-your~ \@@_full_name_env: \
10148     is-unknown. \\
10149     This-error-is-fatal.
10150 }

10151 \@@_msg_new:nn { unknown-column-type-S }
10152 {
10153     Bad-column-type. \\
10154     The-column-type~'S'~in-your~ \@@_full_name_env: \ is-unknown. \\
10155     If-you-want-to-use-the-column-type~'S'~of~siunitx,~you-should~
10156     load-that-package. \\
10157     This-error-is-fatal.
10158 }

10159 \@@_msg_new:nn { tabularnote-forbidden }
10160 {
10161     Forbidden-command. \\
10162     You-can't-use-the-command~ \token_to_str:N \tabularnote \
10163     ~here.~This-command-is-available-only-in~
10164     \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or-in~
10165     the-argument-of-a-command~ \token_to_str:N \caption \ included~
10166     in-an-environment~\{table\}. \\
10167     This-command-will-be-ignored.
10168 }

10169 \@@_msg_new:nn { borders-forbidden }
10170 {
10171     Forbidden-key.\\
10172     You-can't-use-the-key~'borders'~of-the-command~ \token_to_str:N \Block \
10173     because-the-option~'rounded-corners'~
10174     is-in-force-with-a-non-zero-value.\\
10175     This-key-will-be-ignored.
10176 }

10177 \@@_msg_new:nn { bottomrule-without-booktabs }
10178 {
10179     booktabs-not-loaded.\\
10180     You-can't-use-the-key~'tabular/bottomrule'~because-you-haven't~
10181     loaded~'booktabs'.\\
10182     This-key-will-be-ignored.
10183 }

```

```

10184 \@@_msg_new:nn { enumitem~not~loaded }
10185 {
10186   enumitem~not~loaded. \\
10187   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10188   ~because~you~haven't~loaded~'enumitem'. \\
10189   All~the~commands~ \token_to_str:N \tabularnote \ will~be~
10190   ignored~in~the~document.
10191 }
10192 \@@_msg_new:nn { tikz~without~tikz }
10193 {
10194   Tikz~not~loaded. \\
10195   You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
10196   loaded.~If~you~go~on,~that~key~will~be~ignored.
10197 }
10198 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
10199 {
10200   Tikz~not~loaded. \\
10201   You~have~used~the~key~'tikz'~in~the~definition~of~a~
10202   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
10203   You~can~go~on~but~you~will~have~another~error~if~you~actually~
10204   use~that~custom~line.
10205 }
10206 \@@_msg_new:nn { tikz~in~borders~without~tikz }
10207 {
10208   Tikz~not~loaded. \\
10209   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
10210   command~' \token_to_str:N \Block ')~but~tikz~is~not~loaded.~
10211   That~key~will~be~ignored.
10212 }
10213 \@@_msg_new:nn { color~in~custom~line~with~tikz }
10214 {
10215   Erroneous~use.\\
10216   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
10217   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
10218   The~key~'color'~will~be~discarded.
10219 }
10220 \@@_msg_new:nn { Wrong~last~row }
10221 {
10222   Wrong~number.\\
10223   You~have~used~'last~row= \int_use:N \l_@@_last_row_int '~but~your~
10224   \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
10225   If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
10226   last~row.~You~can~avoid~this~problem~by~using~'last~row'~
10227   without~value~(more~compilations~might~be~necessary).
10228 }
10229 \@@_msg_new:nn { Yet~in~env }
10230 {
10231   Nested~environments.\\
10232   Environments~of~nicematrix~can't~be~nested.\\
10233   This~error~is~fatal.
10234 }
10235 \@@_msg_new:nn { Outside~math~mode }
10236 {
10237   Outside~math~mode.\\
10238   The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
10239   (and~not~in~ \token_to_str:N \vcenter ).\\
10240   This~error~is~fatal.
10241 }
10242 \@@_msg_new:nn { One~letter~allowed }
10243 {

```

```

10244     Bad-name.\\
10245     The-value-of-key~' \l_keys_key_str '~must-be-of-length~1~and~
10246     you-have-used~' \l_keys_value_tl '~.\\
10247     It-will-be-ignored.
10248 }
10249 \@@_msg_new:nn { TabularNote~in~CodeAfter }
10250 {
10251     Environment~\{TabularNote\}~forbidden.\\
10252     You-must-use~\{TabularNote\}~at-the-end-of-your~\{NiceTabular\}~
10253     but~*before*~the~ \token_to_str:N \CodeAfter . \\
10254     This-environment~\{TabularNote\}~will-be-ignored.
10255 }
10256 \@@_msg_new:nn { varwidth~not~loaded }
10257 {
10258     varwidth~not~loaded.\\
10259     You-can't-use-the-column-type~'V'~because~'varwidth'~is~not~
10260     loaded.\\
10261     Your~column~will~behave~like~'p'.
10262 }
10263 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
10264 {
10265     Unknown~key.\\
10266     Your~key~' \l_keys_key_str '~is-unknown~for~a~rule.\\
10267     \c_@@_available_keys_str
10268 }
10269 {
10270     The-available-keys-are~(in~alphabetic~order):~
10271     color,~
10272     dotted,~
10273     multiplicity,~
10274     sep-color,~
10275     tikz,~and~total-width.
10276 }
10277
10278 \@@_msg_new:nnn { Unknown~key~for~Block }
10279 {
10280     Unknown~key. \\
10281     The~key~' \l_keys_key_str '~is-unknown~for~the~command~
10282     \token_to_str:N \Block . \\
10283     It-will-be-ignored. \\
10284     \c_@@_available_keys_str
10285 }
10286 {
10287     The-available-keys-are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10288     b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
10289     opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
10290     and~vlines.
10291 }
10292 \@@_msg_new:nnn { Unknown~key~for~Brace }
10293 {
10294     Unknown~key.\\
10295     The~key~' \l_keys_key_str '~is-unknown~for~the~commands~
10296     \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
10297     It-will-be-ignored. \\
10298     \c_@@_available_keys_str
10299 }
10300 {
10301     The-available-keys-are~(in~alphabetic~order):~color,~left-shorten,~
10302     right-shorten,~shorten~(which~fixes~both~left-shorten~and~
10303     right-shorten)~and~yshift.
10304 }

```

```

10305 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10306 {
10307   Unknown~key.\\
10308   The~key~' \l_keys_key_str '~is~unknown.\\
10309   It~will~be~ignored. \\
10310   \c_@@_available_keys_str
10311 }
10312 {
10313   The~available~keys~are~(in~alphabetic~order):~
10314   delimiters/color,~
10315   rules~(with~the~subkeys~'color'~and~'width'),~
10316   sub-matrix~(several~subkeys)~
10317   and~xdots~(several~subkeys).~
10318   The~latter~is~for~the~command~ \token_to_str:N \line .
10319 }
10320 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10321 {
10322   Unknown~key.\\
10323   The~key~' \l_keys_key_str '~is~unknown.\\
10324   It~will~be~ignored. \\
10325   \c_@@_available_keys_str
10326 }
10327 {
10328   The~available~keys~are~(in~alphabetic~order):~
10329   create-cell-nodes,~
10330   delimiters/color~and~
10331   sub-matrix~(several~subkeys).
10332 }
10333 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10334 {
10335   Unknown~key.\\
10336   The~key~' \l_keys_key_str '~is~unknown.\\
10337   That~key~will~be~ignored. \\
10338   \c_@@_available_keys_str
10339 }
10340 {
10341   The~available~keys~are~(in~alphabetic~order):~
10342   'delimiters/color',~
10343   'extra-height',~
10344   'hlines',~
10345   'hvlines',~
10346   'left-xshift',~
10347   'name',~
10348   'right-xshift',~
10349   'rules'~(with~the~subkeys~'color'~and~'width'),~
10350   'slim',~
10351   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10352   and~'right-xshift').\\
10353 }
10354 \@@_msg_new:nnn { Unknown~key~for~notes }
10355 {
10356   Unknown~key.\\
10357   The~key~' \l_keys_key_str '~is~unknown.\\
10358   That~key~will~be~ignored. \\
10359   \c_@@_available_keys_str
10360 }
10361 {
10362   The~available~keys~are~(in~alphabetic~order):~
10363   bottomrule,~
10364   code-after,~
10365   code-before,~
10366   detect-duplicates,~
10367   enumitem-keys,~

```

```

10368     enumitem-keys-para,~
10369     para,~
10370     label-in-list,~
10371     label-in-tabular~and~
10372     style.
10373 }
10374 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10375 {
10376     Unknown~key.\\
10377     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10378     \token_to_str:N \RowStyle . \\
10379     That~key~will~be~ignored. \\
10380     \c_@@_available_keys_str
10381 }
10382 {
10383     The~available~keys~are~(in~alphabetic~order):~
10384     bold,~
10385     cell-space-top-limit,~
10386     cell-space-bottom-limit,~
10387     cell-space-limits,~
10388     color,~
10389     fill~(alias:~rowcolor),~
10390     nb-rows,~
10391     opacity~and~
10392     rounded-corners.
10393 }
10394 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10395 {
10396     Unknown~key.\\
10397     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10398     \token_to_str:N \NiceMatrixOptions . \\
10399     That~key~will~be~ignored. \\
10400     \c_@@_available_keys_str
10401 }
10402 {
10403     The~available~keys~are~(in~alphabetic~order):~
10404     &-in-blocks,~
10405     allow-duplicate-names,~
10406     ampersand-in-blocks,~
10407     caption-above,~
10408     cell-space-bottom-limit,~
10409     cell-space-limits,~
10410     cell-space-top-limit,~
10411     code-for-first-col,~
10412     code-for-first-row,~
10413     code-for-last-col,~
10414     code-for-last-row,~
10415     corners,~
10416     custom-key,~
10417     create-extra-nodes,~
10418     create-medium-nodes,~
10419     create-large-nodes,~
10420     custom-line,~
10421     delimiters~(several~subkeys),~
10422     end-of-row,~
10423     first-col,~
10424     first-row,~
10425     hlines,~
10426     hvlines,~
10427     hvlines-except-borders,~
10428     last-col,~
10429     last-row,~
10430     left-margin,~

```



```

10431 light-syntax,~
10432 light-syntax-expanded,~
10433 matrix/columns-type,~
10434 no-cell-nodes,~
10435 notes~(several~subkeys),~
10436 nullify-dots,~
10437 pgf-node-code,~
10438 renew-dots,~
10439 renew-matrix,~
10440 respect-arraystretch,~
10441 rounded-corners,~
10442 right-margin,~
10443 rules~(with~the~subkeys~'color'~and~'width'),~
10444 small,~
10445 sub-matrix~(several~subkeys),~
10446 vlines,~
10447 xdots~(several~subkeys).
10448 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

10449 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10450 {
10451   Unknown~key.\\
10452   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
10453   \{NiceArray\}. \\
10454   That~key~will~be~ignored. \\
10455   \c_@@_available_keys_str
10456 }
10457 {
10458   The~available~keys~are~(in~alphabetic~order):~
10459   &~in~blocks,~
10460   ampersand~in~blocks,~
10461   b,~
10462   baseline,~
10463   c,~
10464   cell-space-bottom-limit,~
10465   cell-space-limits,~
10466   cell-space-top-limit,~
10467   code~after,~
10468   code~for~first~col,~
10469   code~for~first~row,~
10470   code~for~last~col,~
10471   code~for~last~row,~
10472   columns~width,~
10473   corners,~
10474   create~extra~nodes,~
10475   create~medium~nodes,~
10476   create~large~nodes,~
10477   extra~left~margin,~
10478   extra~right~margin,~
10479   first~col,~
10480   first~row,~
10481   hlines,~
10482   hvlines,~
10483   hvlines~except~borders,~
10484   last~col,~
10485   last~row,~
10486   left~margin,~
10487   light-syntax,~
10488   light-syntax-expanded,~
10489   name,~
10490   no~cell~nodes,~
10491   nullify~dots,~

```

```

10492 pgf-node-code,~
10493 renew-dots,~
10494 respect-arraystretch,~
10495 right-margin,~
10496 rounded-corners,~
10497 rules~(with~the~subkeys~'color'~and~'width'),~
10498 small,~
10499 t,~
10500 vl原因,~
10501 xdots/color,~
10502 xdots/shorten-start,~
10503 xdots/shorten-end,~
10504 xdots/shorten-and~
10505 xdots/line-style.
10506 }

```

This error message is used for the set of keys `nicematrix/NiceMatrix` and `nicematrix/pNiceArray` (but not by `nicematrix/NiceArray` because, for this set of keys, there is no `l` and `r`).

```

10507 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10508 {
10509   Unknown~key.\\
10510   The~key~' \l_keys_key_str 'is~unknown~for~the~
10511   \@@_full_name_env: . \\
10512   That~key~will~be~ignored. \\
10513   \c_@@_available_keys_str
10514 }
10515 {
10516   The~available~keys~are~(in~alphabetic~order):~
10517   &~in~blocks,~
10518   ampersand~in~blocks,~
10519   b,~
10520   baseline,~
10521   c,~
10522   cell-space-bottom-limit,~
10523   cell-space-limits,~
10524   cell-space-top-limit,~
10525   code-after,~
10526   code-for-first-col,~
10527   code-for-first-row,~
10528   code-for-last-col,~
10529   code-for-last-row,~
10530   columns-type,~
10531   columns-width,~
10532   corners,~
10533   create-extra-nodes,~
10534   create-medium-nodes,~
10535   create-large-nodes,~
10536   extra-left-margin,~
10537   extra-right-margin,~
10538   first-col,~
10539   first-row,~
10540   hlines,~
10541   hvlines,~
10542   hvlines-except-borders,~
10543   l,~
10544   last-col,~
10545   last-row,~
10546   left-margin,~
10547   light-syntax,~
10548   light-syntax-expanded,~
10549   name,~
10550   no-cell-nodes,~
10551   nullify-dots,~
10552   pgf-node-code,~

```

```

10553   r,~
10554   renew-dots,~
10555   respect-arraystretch,~
10556   right-margin,~
10557   rounded-corners,~
10558   rules~(with~the~subkeys~'color'~and~'width'),~
10559   small,~
10560   t,~
10561   vl原因,~
10562   xdots/color,~
10563   xdots/shorten-start,~
10564   xdots/shorten-end,~
10565   xdots/shorten-and~
10566   xdots/line-style.
10567 }
10568 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10569 {
10570   Unknown-key.\\
10571   The-key~' \l_keys_key_str '~is-unknown-for-the-environment~
10572   \{NiceTabular\}. \\
10573   That-key-will-be-ignored. \\
10574   \c_@@_available_keys_str
10575 }
10576 {
10577   The-available-keys-are~(in~alphabetic-order):~
10578   &-in-blocks,~
10579   ampersand-in-blocks,~
10580   b,~
10581   baseline,~
10582   c,~
10583   caption,~
10584   cell-space-bottom-limit,~
10585   cell-space-limits,~
10586   cell-space-top-limit,~
10587   code-after,~
10588   code-for-first-col,~
10589   code-for-first-row,~
10590   code-for-last-col,~
10591   code-for-last-row,~
10592   columns-width,~
10593   corners,~
10594   custom-line,~
10595   create-extra-nodes,~
10596   create-medium-nodes,~
10597   create-large-nodes,~
10598   extra-left-margin,~
10599   extra-right-margin,~
10600   first-col,~
10601   first-row,~
10602   hlines,~
10603   hvlines,~
10604   hvlines-except-borders,~
10605   label,~
10606   last-col,~
10607   last-row,~
10608   left-margin,~
10609   light-syntax,~
10610   light-syntax-expanded,~
10611   name,~
10612   no-cell-nodes,~
10613   notes~(several~subkeys),~
10614   nullify-dots,~
10615   pgf-node-code,~

```

```

10616     renew-dots,~
10617     respect-arraystretch,~
10618     right-margin,~
10619     rounded-corners,~
10620     rules~(with~the~subkeys~'color'~and~'width'),~
10621     short-caption,~
10622     t,~
10623     tabularnote,~
10624     vlines,~
10625     xdots/color,~
10626     xdots/shorten-start,~
10627     xdots/shorten-end,~
10628     xdots/shorten-and~
10629     xdots/line-style.
10630 }

10631 \@@_msg_new:nnn { Duplicate~name }
10632 {
10633     Duplicate~name.\\
10634     The~name~' \l_keys_value_tl '~is~already~used~and~you~shouldn't~use~
10635     the~same~environment~name~twice.~You~can~go~on,~but,~
10636     maybe,~you~will~have~incorrect~results~especially~
10637     if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10638     message~again,~use~the~key~'allow-duplicate-names'~in~
10639     ' \token_to_str:N \NiceMatrixOptions '.\\
10640     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10641     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10642 }
10643 {
10644     The~names~already~defined~in~this~document~are:~
10645     \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
10646 }

10647 \@@_msg_new:nn { Option~auto~for~columns-width }
10648 {
10649     Erroneous~use.\\
10650     You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
10651     That~key~will~be~ignored.
10652 }

10653 \@@_msg_new:nn { NiceTabularX~without~X }
10654 {
10655     NiceTabularX~without~X.\\
10656     You~should~not~use~\{NiceTabularX\}~without~X~columns.\\
10657     However,~you~can~go~on.
10658 }

10659 \@@_msg_new:nn { Preamble~forgotten }
10660 {
10661     Preamble~forgotten.\\
10662     You~have~probably~forgotten~the~preamble~of~your~
10663     \@@_full_name_env: . \\
10664     This~error~is~fatal.
10665 }

10666 \@@_msg_new:nn { Invalid~col~number }
10667 {
10668     Invalid~column~number.\\
10669     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
10670     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.
10671 }

10672 \@@_msg_new:nn { Invalid~row~number }
10673 {
10674     Invalid~row~number.\\
10675     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
10676     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.

```

```
10677     }  
10678 \@@_define_com:NNN p  (  )  
10679 \@@_define_com:NNN b  [  ]  
10680 \@@_define_com:NNN v  |  |  
10681 \@@_define_com:NNN V  \l  \l  
10682 \@@_define_com:NNN B  \{  \}
```

Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	3
4	Parameters	9
5	The command <code>\tabularnote</code>	20
6	Command for creation of rectangle nodes	24
7	The options	25
8	Important code used by <code>{NiceArrayWithDelims}</code>	36
9	The <code>\CodeBefore</code>	50
10	The environment <code>{NiceArrayWithDelims}</code>	55
11	Construction of the preamble of the array	60
12	The redefinition of <code>\multicolumn</code>	75
13	The environment <code>{NiceMatrix}</code> and its variants	92
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	93
15	After the construction of the array	95
16	We draw the dotted lines	101
17	The actual instructions for drawing the dotted lines with Tikz	116
18	User commands available in the new environments	122
19	The command <code>\line</code> accessible in code-after	128
20	The command <code>\RowStyle</code>	129
21	Colors of cells, rows and columns	132
22	The vertical and horizontal rules	144
23	The empty corners	161
24	The environment <code>{NiceMatrixBlock}</code>	163
25	The extra nodes	164
26	The blocks	169
27	How to draw the dotted lines transparently	194
28	Automatic arrays	194
29	The redefinition of the command <code>\dotfill</code>	195
30	The command <code>\diagbox</code>	196

31	The keyword <code>\CodeAfter</code>	197
32	The delimiters in the preamble	198
33	The command <code>\SubMatrix</code>	199
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	208
35	The commands <code>HBrace</code> et <code>VBrace</code>	211
36	The command <code>TikzEveryCell</code>	214
37	The command <code>\ShowCellNames</code>	215
38	We process the options at package loading	217
39	About the package underscore	218
40	Error messages of the package	219