

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

August 5, 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 7.2 of `nicematrix`, at the date of 2025/08/05.

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

```
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   {
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
```

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

```
342 \bool_new:N \l_@@_V_of_X_bool
```

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

```
343 \bool_new:N \g_@@_V_of_X_bool
```

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

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

```
345 \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 }`).

```
346 \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.

```
347 \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.

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

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

```
350 \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}`).

```
351 \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`).

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

For `\multicolumn`.

```
353 \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`.

```
354 \tl_new:N \l_@@_columns_type_tl
355 \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 `:`.

```
356 \tl_new:N \l_@@_xdots_down_tl
357 \tl_new:N \l_@@_xdots_up_tl
358 \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`).

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

```
360 \cs_new_protected:Npn \@@_test_if_math_mode:
361 {
362   \if_mode_math: \else:
363     \@@_fatal:n { Outside~math~mode }
364   \fi:
365 }
```

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.

```
366 \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”.

```
367 \colorlet { nicematrix-last-col } { . }
368 \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`).

```
369 \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*.

```
370 \tl_new:N \g_@@_com_or_env_str
371 \tl_gset:Nn \g_@@_com_or_env_str { environment }
```

```
372 \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)`.

```
373 \cs_new:Npn \@@_full_name_env:
374 {
375   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
376     { command \space \c_backslash_str \g_@@_name_env_str }
377     { environment \space \{ \g_@@_name_env_str \} }
378 }
```

```
379 \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.

```
380 \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.

```
381 \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.

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

```
384 \tl_new:N \g_@@_pre_code_after_tl
385 \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.

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

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

```
387 \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.

```
388 \int_new:N \l_@@_old_iRow_int
389 \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).

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

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

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

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

```
392 \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.

```
393 \bool_new:N \l_@@_X_columns_aux_bool
394 \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.

```
395 \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).

```
396 \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).

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

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

```
399 \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).

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

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

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

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

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

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

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

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

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

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

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

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

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

```
411 \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”.

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

```
413 \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.

```
414 \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!).

```
415 \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.

```
416 \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}`.

```
417 \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.

```
418 \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`).

```
419 \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`).

```
420 \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.

```
421 \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).

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

```
423 \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.

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

```
425 \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.

```

426 \dim_new:N \g_@@_delta_x_one_dim
427 \dim_new:N \g_@@_delta_y_one_dim
428 \dim_new:N \g_@@_delta_x_two_dim
429 \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`).

```

430 \int_new:N \l_@@_row_min_int
431 \int_new:N \l_@@_row_max_int
432 \int_new:N \l_@@_col_min_int
433 \int_new:N \l_@@_col_max_int

434 \int_new:N \l_@@_initial_i_int
435 \int_new:N \l_@@_initial_j_int
436 \int_new:N \l_@@_final_i_int
437 \int_new:N \l_@@_final_j_int

```

The following counters will be used when drawing the rules.

```

438 \int_new:N \l_@@_start_int
439 \int_set_eq:NN \l_@@_start_int \c_one_int
440 \int_new:N \l_@@_end_int
441 \int_new:N \l_@@_local_start_int
442 \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.

```

443 \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).

```

444 \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`.

```

445 \tl_new:N \l_@@_fill_tl
446 \tl_new:N \l_@@_opacity_tl
447 \tl_new:N \l_@@_draw_tl
448 \seq_new:N \l_@@_tikz_seq
449 \clist_new:N \l_@@_borders_clist
450 \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.

```

451 \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`.

```

452 \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.

```
453 \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.

```
454 \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`).

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

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

```
459 \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).

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

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

```
461 \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).

```
462 \bool_new:N \l_@@_vlines_block_bool
463 \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.

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

465 \dim_new:N \l_@@_submatrix_extra_height_dim
466 \dim_new:N \l_@@_submatrix_left_xshift_dim
467 \dim_new:N \l_@@_submatrix_right_xshift_dim
468 \clist_new:N \l_@@_hlines_clist
469 \clist_new:N \l_@@_vlines_clist
470 \clist_new:N \l_@@_submatrix_hlines_clist
471 \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}`).

```
472 \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.

```
473 \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`).

```
474 \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.

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

```
477 \int_new:N \l_@@_first_col_int
478 \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).

```
479 \int_new:N \l_@@_last_row_int
480 \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

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

Idem for `\l_@@_last_col_without_value_bool`

```
482 \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.

```
483 \int_new:N \l_@@_last_col_int
484 \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.

```

485 \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`).

```

486 \bool_new:N \l_@@_in_last_col_bool

```

Some utilities

```

487 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
488 {

```

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

```

489 \def \l_tmpa_tl { #1 }
490 \def \l_tmpb_tl { #2 }
491 }

```

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.

```

492 \cs_new_protected:Npn \@@_expand_clist:N #1
493 {
494 \clist_if_in:NnF #1 { all }
495 {
496 \clist_clear:N \l_tmpa_clist
497 \clist_map_inline:Nn #1
498 {

```

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

```

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

```

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

```

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

```

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.

```

511 \hook_gput_code:nnn { begindocument } { . }
512 {
513 \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
514 \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
515 }

```

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.

```
516 \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`.

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

```
521 \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.

```
522 \seq_new:N \l_@@_notes_labels_seq
523 \newcounter { nicematrix_draft }
```

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

```

524 \cs_new_protected:Npn \@@_notes_format:n #1
525 {
526   \setcounter { nicematrix_draft } { #1 }
527   \@@_notes_style:n { nicematrix_draft }
528 }

```

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

```

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

```

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

```

530 \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`.

```

531 \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`.

```

532 \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`).

```

533 \hook_gput_code:nnn { begindocument } { . }
534 {
535   \IfPackageLoadedTF { enumitem }
536   {

```

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.

```

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

```

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`.

```

556     \NewDocumentCommand \tabularnote { o m }
557     {
558       \bool_lazy_or:nnT { \cs_if_exist_p:N \capttype } { \l_@@_in_env_bool }

```

```

559         {
560             \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } { \l_@@_in_env_bool }
561             { \@@_error:n { tabularnote~forbidden } }
562             {
563                 \bool_if:NTF \l_@@_in_caption_bool
564                 \@@_tabularnote_caption:nn
565                 \@@_tabularnote:nn
566                 { #1 } { #2 }
567             }
568         }
569     }
570 }
571 {
572     \NewDocumentCommand \tabularnote { o m }
573     { \@@_err_enumitem_not_loaded: }
574 }
575 }
576 \cs_new_protected:Npn \@@_err_enumitem_not_loaded:
577 {
578     \@@_error_or_warning:n { enumitem~not~loaded }
579     \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
580 }
581 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
582 { \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`.

```

583 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
584 {

```

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).

```

585     \int_zero:N \l_tmpa_int
586     \bool_if:NTF \l_@@_notes_detect_duplicates_bool
587     {

```

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`.

```

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

```

```

602     }
603     \int_if_zero:nT { \l_tmpa_int }
604     {
605         \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
606         \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
607     }
608     \seq_put_right:Ne \l_@@_notes_labels_seq
609     {
610         \tl_if_novalue:nTF { #1 }
611         {
612             \@@_notes_format:n
613             {
614                 \int_eval:n
615                 {
616                     \int_if_zero:nTF { \l_tmpa_int }
617                     { \c@tabularnote }
618                     { \l_tmpa_int }
619                 }
620             }
621         }
622         { #1 }
623     }
624     \peek_meaning:NF \tabularnote
625     {

```

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`.

```

626     \hbox_set:Nn \l_tmpa_box
627     {

```

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

```

628         \@@_notes_label_in_tabular:n
629         {
630             \seq_use:Nnnn
631             \l_@@_notes_labels_seq { , } { , } { , }
632         }
633     }

```

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

```

634     \int_gdecr:N \c@tabularnote
635     \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.

```

636     \int_gincr:N \g_@@_tabularnote_int
637     \refstepcounter { tabularnote }
638     \int_compare:nNnT { \l_tmpa_int } = { \c@tabularnote }
639     { \int_gincr:N \c@tabularnote }
640     \seq_clear:N \l_@@_notes_labels_seq
641     \bool_lazy_or:nnTF
642     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { c } }
643     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { r } }
644     {
645         \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).

```

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

```

```
650 }
```

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`.

```
651 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
652 {
653   \bool_if:NTF \g_@@_caption_finished_bool
654   {
655     \int_compare:nNnT { \c@tabularnote } = { \g_@@_notes_caption_int }
656     { \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`!

```
657     \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
658     { \@@_error:n { Identical~notes~in~caption } }
659   }
660 }
```

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

```
661     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
662     {
```

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.

```
663       \bool_gset_true:N \g_@@_caption_finished_bool
664       \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
665       \int_gzero:N \c@tabularnote
666     }
667     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
668 }
```

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!

```
669   \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
670   \seq_put_right:Ne \l_@@_notes_labels_seq
671   {
672     \tl_if_novalue:nTF { #1 }
673     { \@@_notes_format:n { \int_use:N \c@tabularnote } }
674     { #1 }
675   }
676   \peek_meaning:NF \tabularnote
677   {
678     \@@_notes_label_in_tabular:n
679     { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
680     \seq_clear:N \l_@@_notes_labels_seq
681   }
682 }

683 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
684 { \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.

```

685 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
686 {
687   \begin { pgfscope }
688   \pgfset
689   {
690     inner~sep = \c_zero_dim ,
691     minimum~size = \c_zero_dim
692   }
693   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
694   \pgfnode
695   { rectangle }
696   { center }
697   {
698     \vbox_to_ht:nn
699     { \dim_abs:n { #5 - #3 } }
700     {
701       \vfill
702       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
703     }
704   }
705   { #1 }
706   { }
707   \end { pgfscope }
708 }

```

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.

```

709 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
710 {
711   \begin { pgfscope }
712   \pgfset
713   {
714     inner~sep = \c_zero_dim ,
715     minimum~size = \c_zero_dim
716   }
717   \pgftransformshift { \pgfpoint scale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
718   \pgfpointdiff { #3 } { #2 }
719   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
720   \pgfnode
721   { rectangle }
722   { center }
723   {
724     \vbox_to_ht:nn
725     { \dim_abs:n \l_tmpb_dim }
726     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
727   }
728   { #1 }
729   { }
730   \end { pgfscope }
731 }

```

7 The options

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

```

732 \tl_new:N \l_@@_caption_tl

```

```

733 \tl_new:N \l_@@_short_caption_tl
734 \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).

```

735 \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`.

```

736 \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`).

```

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

```

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

```

739 \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.

```

740 \dim_new:N \l_@@_xdots_inter_dim
741 \hook_gput_code:nnn { begindocument } { . }
742 { \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).

```

743 \dim_new:N \l_@@_xdots_shorten_start_dim
744 \dim_new:N \l_@@_xdots_shorten_end_dim
745 \hook_gput_code:nnn { begindocument } { . }
746 {
747   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
748   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
749 }

```

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.

```

750 \dim_new:N \l_@@_xdots_radius_dim
751 \hook_gput_code:nnn { begindocument } { . }
752 { \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.

```

753 \tl_new:N \l_@@_xdots_line_style_tl
754 \tl_const:Nn \c_@@_standard_tl { standard }
755 \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`.

```
756 \bool_new:N \l_@@_light_syntax_bool
757 \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).

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

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

```
760 \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`).

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

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

```
762 \bool_new:N \l_@@_parallelize_diags_bool
763 \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.

```
764 \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.).

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

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

```
766 \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).

```
767 \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!

```
768 \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.

```
769 \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”.

```
770 \bool_new:N \l_@@_medium_nodes_bool
771 \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).

```
772 \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`).

```
773 \dim_new:N \l_@@_left_margin_dim
774 \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`.

```
775 \dim_new:N \l_@@_extra_left_margin_dim
776 \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.

```
777 \tl_new:N \l_@@_end_of_row_tl
778 \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 “:”.

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

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

```
780 \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.

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

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

```

805     {
806       \bool_lazy_or:nnTF
807       { \cs_if_exist_p:N \tikzpicture }
808       { \str_if_eq_p:nn { #1 } { standard } }
809       { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
810       { \@@_error:n { bad-option-for-line-style } }
811     } ,
812     line-style .value_required:n = true ,
813     color .tl_set:N = \l_@@_xdots_color_tl ,
814     color .value_required:n = true ,
815     radius .code:n =
816       \hook_gput_code:nnn { begindocument } { . }
817       { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
818     radius .value_required:n = true ,
819     inter .code:n =
820       \hook_gput_code:nnn { begindocument } { . }
821       { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
822     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}`.

```

823     down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
824     up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
825     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).

```

826     draw-first .code:n = \prg_do_nothing: ,
827     unknown .code:n = \@@_error:n { Unknown-key-for-xdots }
828   }

```

```

829 \keys_define:nn { nicematrix / rules }
830 {
831   color .tl_set:N = \l_@@_rules_color_tl ,
832   color .value_required:n = true ,
833   width .dim_set:N = \arrayrulewidth ,
834   width .value_required:n = true ,
835   unknown .code:n = \@@_error:n { Unknown-key-for-rules }
836 }
837 \cs_new_protected:Npn \@@_err_key_color_inside:
838 {
839   \@@_error_or_warning:n { key-color-inside }
840   \cs_gset:Npn \@@_err_key_color_inside: { }
841 }

```

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

```

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

```

```

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

```

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

```

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

```

```

917 \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
918 \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
919 \bool_set_true:N \l_@@_hvlines_bool
920 \bool_set_true:N \l_@@_except_borders_bool
921 } ,
922 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.

```

923 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
924 renew-dots .value_forbidden:n = true ,
925 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
926 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
927 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
928 create-extra-nodes .meta:n =
929   { create-medium-nodes , create-large-nodes } ,
930 left-margin .dim_set:N = \l_@@_left_margin_dim ,
931 left-margin .default:n = \arraycolsep ,
932 right-margin .dim_set:N = \l_@@_right_margin_dim ,
933 right-margin .default:n = \arraycolsep ,
934 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
935 margin .default:n = \arraycolsep ,
936 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
937 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
938 extra-margin .meta:n =
939   { extra-left-margin = #1 , extra-right-margin = #1 } ,
940 extra-margin .value_required:n = true ,
941 respect-arraystretch .code:n =
942   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
943 respect-arraystretch .value_forbidden:n = true ,
944 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
945 pgf-node-code .value_required:n = true
946 }

```

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

```

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

```

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

```

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

```

```

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

```

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

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

984 \keys_define:nn { nicematrix / notes }
985 {
986     para .bool_set:N = \l_@@_notes_para_bool ,
987     para .default:n = true ,
988     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
989     code-before .value_required:n = true ,
990     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
991     code-after .value_required:n = true ,
992     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
993     bottomrule .default:n = true ,
994     style .cs_set:Np = \@@_notes_style:n #1 ,
995     style .value_required:n = true ,
996     label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
997     label-in-tabular .value_required:n = true ,
998     label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
999     label-in-list .value_required:n = true ,
1000     enumitem-keys .code:n =
1001     {
1002         \hook_gput_code:nnn { begindocument } { . }
1003         {
1004             \IfPackageLoadedT { enumitem }
1005             { \setlist* [ tabularnotes ] { #1 } }
1006         }
1007     } ,
1008     enumitem-keys .value_required:n = true ,
1009     enumitem-keys-para .code:n =
1010     {
1011         \hook_gput_code:nnn { begindocument } { . }
1012         {
1013             \IfPackageLoadedT { enumitem }
1014             { \setlist* [ tabularnotes* ] { #1 } }
1015         }
1016     } ,
1017     enumitem-keys-para .value_required:n = true ,
1018     detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1019     detect-duplicates .default:n = true ,
1020     unknown .code:n = \@@_error:n { Unknown~key~for~notes }
1021 }

1022 \keys_define:nn { nicematrix / delimiters }
1023 {
1024     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1025     max-width .default:n = true ,
1026     color .tl_set:N = \l_@@_delimiters_color_tl ,

```



```

1027     color .value_required:n = true ,
1028 }

```

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

```

1029 \keys_define:nn { nicematrix }
1030 {
1031   NiceMatrixOptions .inherit:n =
1032     { nicematrix / Global } ,
1033   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1034   NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1035   NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1036   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1037   SubMatrix / rules .inherit:n = nicematrix / rules ,
1038   CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1039   CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1040   CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1041   NiceMatrix .inherit:n =
1042     {
1043       nicematrix / Global ,
1044       nicematrix / environments ,
1045     } ,
1046   NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1047   NiceMatrix / rules .inherit:n = nicematrix / rules ,
1048   NiceTabular .inherit:n =
1049     {
1050       nicematrix / Global ,
1051       nicematrix / environments
1052     } ,
1053   NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1054   NiceTabular / rules .inherit:n = nicematrix / rules ,
1055   NiceTabular / notes .inherit:n = nicematrix / notes ,
1056   NiceArray .inherit:n =
1057     {
1058       nicematrix / Global ,
1059       nicematrix / environments ,
1060     } ,
1061   NiceArray / xdots .inherit:n = nicematrix / xdots ,
1062   NiceArray / rules .inherit:n = nicematrix / rules ,
1063   pNiceArray .inherit:n =
1064     {
1065       nicematrix / Global ,
1066       nicematrix / environments ,
1067     } ,
1068   pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1069   pNiceArray / rules .inherit:n = nicematrix / rules ,
1070 }

```

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

```

1071 \keys_define:nn { nicematrix / NiceMatrixOptions }
1072 {
1073   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1074   delimiters / color .value_required:n = true ,
1075   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1076   delimiters / max-width .default:n = true ,
1077   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1078   delimiters .value_required:n = true ,
1079   width .dim_set:N = \l_@@_width_dim ,
1080   width .value_required:n = true ,
1081   last-col .code:n =
1082     \tl_if_empty:nF { #1 }

```

```

1083     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1084     \int_zero:N \l_@@_last_col_int ,
1085     small .bool_set:N = \l_@@_small_bool ,
1086     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.

```

1087     renew-matrix .code:n = \@@_renew_matrix: ,
1088     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}`.

```

1089     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.

```

1090     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)`.

```

1091     \str_if_eq:eeTF { #1 } { auto }
1092     { \@@_error:n { Option-auto-for-columns-width } }
1093     { \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.

```

1094     allow-duplicate-names .code:n =
1095     \cs_set:Nn \@@_err_duplicate_names:n { } ,
1096     allow-duplicate-names .value_forbidden:n = true ,
1097     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1098     notes .value_required:n = true ,
1099     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1100     sub-matrix .value_required:n = true ,
1101     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1102     matrix / columns-type .value_required:n = true ,
1103     caption-above .bool_set:N = \l_@@_caption_above_bool ,
1104     caption-above .default:n = true ,
1105     unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrixOptions }
1106 }

```

`\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.

```

1107 \NewDocumentCommand \NiceMatrixOptions { m }
1108 { \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.

```

1109 \keys_define:nn { nicematrix / NiceMatrix }
1110 {
1111     last-col .code:n = \tl_if_empty:nTF { #1 }
1112     {
1113         \bool_set_true:N \l_@@_last_col_without_value_bool
1114         \int_set:Nn \l_@@_last_col_int { -1 }
1115     }
1116     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1117     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1118     columns-type .value_required:n = true ,
1119     l .meta:n = { columns-type = l } ,
1120     r .meta:n = { columns-type = r } ,

```

```

1121 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1122 delimiters / color .value_required:n = true ,
1123 delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1124 delimiters / max-width .default:n = true ,
1125 delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1126 delimiters .value_required:n = true ,
1127 small .bool_set:N = \l_@@_small_bool ,
1128 small .value_forbidden:n = true ,
1129 unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1130 }

```

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

```

1131 \keys_define:nn { nicematrix / NiceArray }
1132 {

```

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.

```

1133 small .bool_set:N = \l_@@_small_bool ,
1134 small .value_forbidden:n = true ,
1135 last-col .code:n = \tl_if_empty:nF { #1 }
1136 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1137 \int_zero:N \l_@@_last_col_int ,
1138 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1139 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1140 unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1141 }
1142 \keys_define:nn { nicematrix / pNiceArray }
1143 {
1144 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1145 last-col .code:n = \tl_if_empty:nF { #1 }
1146 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1147 \int_zero:N \l_@@_last_col_int ,
1148 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1149 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1150 delimiters / color .value_required:n = true ,
1151 delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1152 delimiters / max-width .default:n = true ,
1153 delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1154 delimiters .value_required:n = true ,
1155 small .bool_set:N = \l_@@_small_bool ,
1156 small .value_forbidden:n = true ,
1157 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1158 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1159 unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1160 }

```

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

```

1161 \keys_define:nn { nicematrix / NiceTabular }
1162 {

```

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.

```

1163 width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1164 \bool_set_true:N \l_@@_width_used_bool ,
1165 width .value_required:n = true ,
1166 notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1167 tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1168 tabularnote .value_required:n = true ,
1169 caption .tl_set:N = \l_@@_caption_tl ,

```

```

1170   caption .value_required:n = true ,
1171   short-caption .tl_set:N = \l_@@_short_caption_tl ,
1172   short-caption .value_required:n = true ,
1173   label .tl_set:N = \l_@@_label_tl ,
1174   label .value_required:n = true ,
1175   last-col .code:n = \tl_if_empty:nF { #1 }
1176     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1177     \int_zero:N \l_@@_last_col_int ,
1178   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1179   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1180   unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
1181 }

```

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
```

```

1182 \keys_define:nn { nicematrix / CodeAfter }
1183 {
1184   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1185   delimiters / color .value_required:n = true ,
1186   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1187   rules .value_required:n = true ,
1188   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1189   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1190   sub-matrix .value_required:n = true ,
1191   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1192 }

```

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}`).

```

1193 \cs_new_protected:Npn \@@_cell_begin:
1194 {

```

`\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.

```
1195   \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).

```
1196   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1197   \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.

```

1198   \int_compare:nNnT { \c@jCol } = { \c_one_int }
1199   {
1200     \int_compare:nNnT { \l_@@_first_col_int } = { \c_one_int }
1201     { \@@_begin_of_row: }
1202   }

```

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`:

```
1203 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1204 \@@_tuning_not_tabular_begin:
1205 \@@_tuning_first_row:
1206 \@@_tuning_last_row:
1207 \g_@@_row_style_tl
1208 }
```

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.

```
1209 \cs_new_protected:Npn \@@_tuning_first_row:
1210 {
1211   \if_int_compare:w \c@iRow = \c_zero_int
1212     \if_int_compare:w \c@jCol > \c_zero_int
1213       \l_@@_code_for_first_row_tl
1214       \xglobal \colorlet { nicematrix-first-row } { . }
1215     \fi:
1216   \fi:
1217 }
```

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.

```
1218 \cs_new_protected:Npn \@@_tuning_last_row:
1219 {
1220   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1221     \l_@@_code_for_last_row_tl
1222     \xglobal \colorlet { nicematrix-last-row } { . }
1223   \fi:
1224 }
```

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

```
1225 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```

1226 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1227 {
1228   \m@th
1229   \c_math_toggle_token

```

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

```

1230   \@@_tuning_key_small:
1231 }
1232 \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.

```

1233 \cs_new_protected:Npn \@@_begin_of_row:
1234 {
1235   \int_gincr:N \c@iRow
1236   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1237   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1238   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1239   \pgfpicture
1240   \pgfrememberpicturepositiononpagetrue
1241   \pgfcoordinate
1242   { \@@_env: - row - \int_use:N \c@iRow - base }
1243   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1244   \str_if_empty:NF \l_@@_name_str
1245   {
1246     \pgfnodealias
1247     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1248     { \@@_env: - row - \int_use:N \c@iRow - base }
1249   }
1250   \endpgfpicture
1251 }

```

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.

```

1252 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1253 {
1254   \int_if_zero:nTF { \c@iRow }
1255   {
1256     \dim_compare:nNnT
1257     { \box_dp:N \l_@@_cell_box } > { \g_@@_dp_row_zero_dim }
1258     { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1259     \dim_compare:nNnT
1260     { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_zero_dim }
1261     { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1262   }
1263   {
1264     \int_compare:nNnT { \c@iRow } = { \c_one_int }
1265     {
1266       \dim_compare:nNnT
1267       { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_one_dim }
1268       { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1269     }
1270   }
1271 }

```

```

1272 \cs_new_protected:Npn \@@_rotate_cell_box:
1273 {
1274   \box_rotate:Nn \l_@@_cell_box { 90 }
1275   \bool_if:NTF \g_@@_rotate_c_bool
1276   {
1277     \hbox_set:Nn \l_@@_cell_box
1278     {
1279       \m@th
1280       \c_math_toggle_token
1281       \vcenter { \box_use:N \l_@@_cell_box }
1282       \c_math_toggle_token
1283     }
1284   }
1285   {
1286     \int_compare:nNtT { \c@iRow } = { \l_@@_last_row_int }
1287     {
1288       \vbox_set_top:Nn \l_@@_cell_box
1289       {
1290         \vbox_to_zero:n { }
1291         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1292         \box_use:N \l_@@_cell_box
1293       }
1294     }
1295   }
1296   \bool_gset_false:N \g_@@_rotate_bool
1297   \bool_gset_false:N \g_@@_rotate_c_bool
1298 }
1299 \cs_new_protected:Npn \@@_adjust_size_box:
1300 {
1301   \dim_compare:nNtT { \g_@@_blocks_wd_dim } > { \c_zero_dim }
1302   {
1303     \box_set_wd:Nn \l_@@_cell_box
1304     { \dim_max:nn { \box_wd:N \l_@@_cell_box } { \g_@@_blocks_wd_dim } }
1305     \dim_gzero:N \g_@@_blocks_wd_dim
1306   }
1307   \dim_compare:nNtT { \g_@@_blocks_dp_dim } > { \c_zero_dim }
1308   {
1309     \box_set_dp:Nn \l_@@_cell_box
1310     { \dim_max:nn { \box_dp:N \l_@@_cell_box } { \g_@@_blocks_dp_dim } }
1311     \dim_gzero:N \g_@@_blocks_dp_dim
1312   }
1313   \dim_compare:nNtT { \g_@@_blocks_ht_dim } > { \c_zero_dim }
1314   {
1315     \box_set_ht:Nn \l_@@_cell_box
1316     { \dim_max:nn { \box_ht:N \l_@@_cell_box } { \g_@@_blocks_ht_dim } }
1317     \dim_gzero:N \g_@@_blocks_ht_dim
1318   }
1319 }
1320 \cs_new_protected:Npn \@@_cell_end:
1321 {

```

The following command is nullified in the tabulars.

```

1322   \@@_tuning_not_tabular_end:
1323   \hbox_set_end:
1324   \@@_cell_end_i:
1325 }
1326 \cs_new_protected:Npn \@@_cell_end_i:
1327 {

```

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.

```

1328   \g_@@_cell_after_hook_tl
1329   \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }

```

```

1330 \@@_adjust_size_box:
1331 \box_set_ht:Nn \l_@@_cell_box
1332 { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1333 \box_set_dp:Nn \l_@@_cell_box
1334 { \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”).

```

1335 \@@_update_max_cell_width:

```

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

```

1336 \@@_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.

```

1337 \bool_if:NTF \g_@@_empty_cell_bool
1338 { \box_use_drop:N \l_@@_cell_box }
1339 {
1340   \bool_if:NTF \g_@@_not_empty_cell_bool
1341   { \@@_print_node_cell: }
1342   {
1343     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
1344     { \@@_print_node_cell: }
1345     { \box_use_drop:N \l_@@_cell_box }
1346   }
1347 }
1348 \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
1349 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1350 \bool_gset_false:N \g_@@_empty_cell_bool
1351 \bool_gset_false:N \g_@@_not_empty_cell_bool
1352 }

```

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

```

1353 \cs_new_protected:Npn \@@_update_max_cell_width:
1354 {
1355   \dim_gset:Nn \g_@@_max_cell_width_dim
1356   { \dim_max:nn { \g_@@_max_cell_width_dim } { \box_wd:N \l_@@_cell_box } }
1357 }

```

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`).

```

1358 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1359 {

```



```

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

1372 \pgfset
1373 {
1374   nicematrix / cell-node /.style =
1375   {
1376     inner-sep = \c_zero_dim ,
1377     minimum-width = \c_zero_dim
1378   }
1379 }

```

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.

```

1380 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1381 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1382 {
1383   \use:c
1384   {
1385     __siunitx_table_align_
1386     \bool_if:NTF \l__siunitx_table_text_bool
1387     { \l__siunitx_table_align_text_tl }
1388     { \l__siunitx_table_align_number_tl }
1389   }
1390   :n
1391   { #1 }
1392 }

```

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!

```

1393 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1394 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1395 {
1396   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1397   \hbox:n
1398   {
1399     \pgfsys@markposition
1400     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1401   }
1402   #1
1403   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1404   \hbox:n
1405   {
1406     \pgfsys@markposition
1407     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1408   }
1409 }

```

```

1410 \cs_new_protected:Npn \@@_print_node_cell:
1411 {
1412   \socket_use:nn { nicematrix / siunitx-wrap }
1413   { \socket_use:nn { nicematrix / create-cell-nodes } \@@_node_cell: }
1414 }

```

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

```

1415 \cs_new_protected:Npn \@@_node_cell:
1416 {
1417   \pgfpicture
1418   \pgfsetbaseline \c_zero_dim
1419   \pgfrememberpicturepositiononpagetrue
1420   \pgfset { nicematrix / cell-node }
1421   \pgfnode
1422   { rectangle }
1423   { base }
1424   {

```

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).

```

1425     \set@color
1426     \box_use:N \l_@@_cell_box
1427   }
1428   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1429   { \l_@@_pgf_node_code_tl }
1430   \str_if_empty:NF \l_@@_name_str
1431   {
1432     \pgfnodealias
1433     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1434     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1435   }
1436   \endpgfpicture
1437 }

```

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).

```

1438 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1439 {
1440   \bool_if:nTF { #1 } { \tl_gput_left:ce } { \tl_gput_right:ce }
1441   { g_@@_ #2 _ lines _ tl }
1442   {
1443     \use:c { @@ _ draw _ #2 : nnn }
1444     { \int_use:N \c@iRow }

```

```

1445         { \int_use:N \c@jCol }
1446         { \exp_not:n { #3 } }
1447     }
1448 }

1449 \cs_new_protected:Npn \@@_array:n
1450 {
1451     \dim_set:Nn \col@sep
1452     { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1453     \dim_compare:nNnTF { \l_@@_tabular_width_dim } = { \c_zero_dim }
1454     { \def \@halignto { } }
1455     { \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`.

```

1456     \@tabarray

```

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

```

1457     [ \str_if_eq:eeTF \l_@@_baseline_tl { c } { c } { t } ]
1458 }
1459 \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`.

```

1460 \bool_if:nTF
1461 { \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`.

```

1462 { \cs_set_eq:cN { @@_old_ar@ialign: } \ar@ialign }
1463 { \cs_set_eq:NN \@@_old_ialign: \ialign }

```

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

```

1464 \cs_new_protected:Npn \@@_create_row_node:
1465 {
1466     \int_compare:nNnT { \c@iRow } > { \g_@@_last_row_node_int }
1467     {
1468         \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1469         \@@_create_row_node_i:
1470     }
1471 }

1472 \cs_new_protected:Npn \@@_create_row_node_i:
1473 {

```

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

```

1474     \hbox
1475     {
1476         \bool_if:NT \l_@@_code_before_bool
1477         {
1478             \vtop
1479             {
1480                 \skip_vertical:N 0.5\arrayrulewidth
1481                 \pgfsys@markposition
1482                 { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1483                 \skip_vertical:N -0.5\arrayrulewidth
1484             }
1485         }
1486         \pgfpicture
1487         \pgfrememberpicturepositiononpagetrue

```

```

1488     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1489     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1490     \str_if_empty:NF \l_@@_name_str
1491     {
1492         \pgfnodealias
1493         { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1494         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1495     }
1496     \endpgfpicture
1497 }
1498 }

1499 \cs_new_protected:Npn \@@_in_everycr:
1500 {
1501     \bool_if:NT \c_@@_recent_array_bool
1502     {
1503         \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1504         \tbl_update_cell_data_for_next_row:
1505     }
1506     \int_gzero:N \c@jCol
1507     \bool_gset_false:N \g_@@_after_col_zero_bool
1508     \bool_if:NF \g_@@_row_of_col_done_bool
1509     {
1510         \@@_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).

```

1511     \clist_if_empty:NF \l_@@_hlines_clist
1512     {
1513         \str_if_eq:eeF \l_@@_hlines_clist { all }
1514         {
1515             \clist_if_in:NeT
1516             \l_@@_hlines_clist
1517             { \int_eval:n { \c@iRow + 1 } }
1518         }
1519     }

```

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.

```

1520         \int_compare:nNnT { \c@iRow } > { -1 }
1521         {
1522             \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
1523             { \hrule height \arrayrulewidth width \c_zero_dim }
1524         }
1525     }
1526 }
1527 }
1528 }

```

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

```

1529 \cs_set_protected:Npn \@@_renew_dots:
1530 {
1531     \cs_set_eq:NN \ldots \@@_Ldots:
1532     \cs_set_eq:NN \cdots \@@_Cdots:
1533     \cs_set_eq:NN \vdots \@@_Vdots:
1534     \cs_set_eq:NN \ddots \@@_Ddots:
1535     \cs_set_eq:NN \iddots \@@_Iddots:
1536     \cs_set_eq:NN \dots \@@_Ldots:
1537     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1538 }

```

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.

```

1539 \hook_gput_code:nnn { begindocument } { . }
1540 {
1541   \IfPackageLoadedTF { booktabs }
1542   {
1543     \cs_new_protected:Npn \@_patch_booktabs:
1544       { \tl_put_left:Nn \@BTnormal \@_create_row_node_i: }
1545   }
1546   { \cs_new_protected:Npn \@_patch_booktabs: { } }
1547 }

```

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`.

```

1548 \cs_new_protected:Npn \@_some_initialization:
1549 {
1550   \@_everycr:
1551   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1552   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1553   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1554   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1555   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1556   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1557 }

```

```

1558 \cs_new_protected:Npn \@_pre_array_ii:
1559 {

```

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

```

1560   \fp_gzero:N \g_@@_total_X_weight_fp
1561   \bool_gset_false:N \g_@@_V_of_X_bool
1562   \@_expand_clist:N \l_@@_hlines_clist
1563   \@_expand_clist:N \l_@@_vlines_clist
1564   \@_patch_booktabs:
1565   \box_clear_new:N \l_@@_cell_box
1566   \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}`).

```

1567   \bool_if:NT \l_@@_small_bool
1568   {
1569     \def \arraystretch { 0.47 }
1570     \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.

```

1571     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1572 }

```

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!

```

1573     \bool_if:NT \g_@@_create_cell_nodes_bool
1574     {
1575         \tl_put_right:Nn \@@_begin_of_row:
1576         {
1577             \pgfsys@markposition
1578             { \@@_env: - row - \int_use:N \c@iRow - base }
1579         }
1580         \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1581     }

```

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`.

```

1582     \bool_if:NT \c_@@_recent_array_bool
1583     {
1584         \bool_if:NF \c_@@_revtex_bool
1585         {
1586             \def \ar@ialign
1587             {
1588                 \bool_if:NT \c_@@_testphase_table_bool
1589                 \tbl_init_cell_data_for_table:
1590                 \@@_some_initialization:
1591                 \dim_zero:N \tabskip

```

After its first use, the definition of `\ar@ialign` will revert automatically to its default definition. With this programming, we will have, in the cells of the array, a clean version of `\ar@ialign`. We use `\cs_set_eq:Nc` instead of `\cs_set_eq:NN` in order to avoid a message when `explcheck` is used on `nicematrix.sty`.

```

1592                 \cs_set_eq:Nc \ar@ialign { @@_old_ar@ialign: }
1593                 \halign
1594             }
1595         }
1596     }

```

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`.

```

1597     {
1598         \def \ialign
1599         {
1600             \@@_some_initialization:
1601             \dim_zero:N \tabskip
1602             \cs_set_eq:NN \ialign \@@_old_ialign:
1603             \halign
1604         }
1605     }

```

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).

```

1606     \bool_if:NT \c_@@_revtex_bool
1607     {

```

```

1608     \IfPackageLoadedT { colortbl }
1609     { \cs_set_protected:Npn \CT@setup { } }
1610 }

```

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).

```

1611 \cs_set_eq:NN \@@_old_ldots: \ldots
1612 \cs_set_eq:NN \@@_old_cdots: \cdots
1613 \cs_set_eq:NN \@@_old_vdots: \vdots
1614 \cs_set_eq:NN \@@_old_ddots: \ddots
1615 \cs_set_eq:NN \@@_old_iddots: \iddots
1616 \bool_if:NTF \l_@@_standard_cline_bool
1617 { \cs_set_eq:NN \cline \@@_standard_cline: }
1618 { \cs_set_eq:NN \cline \@@_cline: }
1619 \cs_set_eq:NN \Ldots \@@_Ldots:
1620 \cs_set_eq:NN \Cdots \@@_Cdots:
1621 \cs_set_eq:NN \Vdots \@@_Vdots:
1622 \cs_set_eq:NN \Ddots \@@_Ddots:
1623 \cs_set_eq:NN \Iddots \@@_Iddots:
1624 \cs_set_eq:NN \Hline \@@_Hline:
1625 \cs_set_eq:NN \Hspace \@@_Hspace:
1626 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1627 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1628 \cs_set_eq:NN \Block \@@_Block:
1629 \cs_set_eq:NN \rotate \@@_rotate:
1630 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1631 \cs_set_eq:NN \dotfill \@@_dotfill:
1632 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1633 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1634 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1635 \cs_set_eq:NN \TopRule \@@_TopRule
1636 \cs_set_eq:NN \MidRule \@@_MidRule
1637 \cs_set_eq:NN \BottomRule \@@_BottomRule
1638 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1639 \cs_set_eq:NN \Hbrace \@@_Hbrace
1640 \cs_set_eq:NN \Vbrace \@@_Vbrace
1641 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1642 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1643 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1644 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1645 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1646 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1647 \int_compare:nNt { \l_@@_first_row_int } > { \c_zero_int }
1648 { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1649 \int_compare:nNt { \l_@@_last_row_int } < { \c_zero_int }
1650 { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1651 \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:.`

```

1652 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1653 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1654 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1655 \@@_revert_colortbl:

```

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

```

1656 \tl_if_exist:NT \l_@@_note_in_caption_tl
1657 {

```

```

1658     \tl_if_empty:NF \l_@@_note_in_caption_tl
1659     {
1660         \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1661         \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1662     }
1663 }

```

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).

```

1664     \seq_gclear:N \g_@@_multicolumn_cells_seq
1665     \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).

```

1666     \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`).

```

1667     \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.

```

1668     \int_gzero_new:N \g_@@_col_total_int
1669     \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1670     \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.

```

1671     \tl_gclear_new:N \g_@@_Cdots_lines_tl
1672     \tl_gclear_new:N \g_@@_Ldots_lines_tl
1673     \tl_gclear_new:N \g_@@_Vdots_lines_tl
1674     \tl_gclear_new:N \g_@@_Ddots_lines_tl
1675     \tl_gclear_new:N \g_@@_Iddots_lines_tl
1676     \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1677     \tl_gclear:N \g_nicematrix_code_before_tl
1678     \tl_gclear:N \g_@@_pre_code_before_tl
1679 }

```

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

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

```

1680 \cs_new_protected:Npn \@@_pre_array:
1681 {
1682     \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1683     \int_gzero_new:N \c@iRow
1684     \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1685     \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).

```

1686     \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
1687     {
1688         \bool_set_true:N \l_@@_last_row_without_value_bool

```



```

1689     \bool_if:NT \g_@@_aux_found_bool
1690     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1691   }
1692   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
1693   {
1694     \bool_if:NT \g_@@_aux_found_bool
1695     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1696   }

```

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”.

```

1697   \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
1698   {
1699     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1700     {
1701       \dim_compare:nNnT { \g_@@_ht_last_row_dim } < { \box_ht:N \l_@@_cell_box }
1702       { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1703       \dim_compare:nNnT { \g_@@_dp_last_row_dim } < { \box_dp:N \l_@@_cell_box }
1704       { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1705     }
1706   }

1707   \seq_gclear:N \g_@@_cols_vlism_seq
1708   \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1709   \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.

```

1710   \seq_gset_eq:NN \g_@@_pos_of_blocks_seq \g_@@_future_pos_of_blocks_seq
1711   \seq_gclear:N \g_@@_future_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

1712   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1713   \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.

```

1714   \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1715   \@@_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.

```

1716   \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]`).

```

1717   \dim_zero_new:N \l_@@_left_delim_dim
1718   \dim_zero_new:N \l_@@_right_delim_dim
1719   \bool_if:NTF \g_@@_delims_bool
1720   {

```

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

```

1721     \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1722     \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1723     \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1724     \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1725   }
1726   {
1727     \dim_gset:Nn \l_@@_left_delim_dim
1728       { 2 \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1729     \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1730   }

```

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).

```

1731   \hbox_set:Nw \l_@@_the_array_box
1732   \skip_horizontal:N \l_@@_left_margin_dim
1733   \skip_horizontal:N \l_@@_extra_left_margin_dim
1734   \bool_if:NT \c_@@_recent_array_bool
1735     { \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).

```

1736   \m@th
1737   \c_math_toggle_token
1738   \bool_if:NTF \l_@@_light_syntax_bool
1739     { \use:c { @@-light-syntax } }
1740     { \use:c { @@-normal-syntax } }
1741 }

```

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

```

1742 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1743 {
1744   \tl_set:Nn \l_tmpa_tl { #1 }
1745   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1746     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1747   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1748   \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...

```

1749   \@@_pre_array:
1750 }

```

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).

```

1751 \cs_new_protected:Npn \@@_pre_code_before:
1752 {

```

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).

```

1753 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1754 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1755 \int_set:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1756 \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.

```

1757 \pgfsys@markposition { \@@_env: - position }
1758 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1759 \pgfpicture
1760 \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1761 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int + 1 }
1762 {
1763   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1764   \pgfcoordinate { \@@_env: - row - ##1 }
1765   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1766 }

```

Now, the recreation of the `col` nodes.

```

1767 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int + 1 }
1768 {
1769   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1770   \pgfcoordinate { \@@_env: - col - ##1 }
1771   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1772 }

```

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

```

1773 \@@_create_diag_nodes:

```

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

```

1774 \bool_if:NT \g_@@_create_cell_nodes_bool { \@@_recreate_cell_nodes: }
1775 \endpgfpicture

```

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

```

1776 \@@_create_blocks_nodes:
1777 \IfPackageLoadedT { tikz }
1778 {
1779   \tikzset
1780   {
1781     every-picture / .style =
1782     { overlay , name~prefix = \@@_env: - }
1783   }
1784 }
1785 \cs_set_eq:NN \cellcolor \@@_cellcolor
1786 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1787 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1788 \cs_set_eq:NN \rowcolor \@@_rowcolor
1789 \cs_set_eq:NN \rowcolors \@@_rowcolors
1790 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1791 \cs_set_eq:NN \arraycolor \@@_arraycolor
1792 \cs_set_eq:NN \columncolor \@@_columncolor
1793 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1794 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1795 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1796 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1797 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n

```

```

1798 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1799 }

```

```

1800 \cs_new_protected:Npn \@@_exec_code_before:
1801 {

```

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...

```

1802 \clist_map_inline:Nn \l_@@_corners_cells_clist
1803 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1804 \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`.

```

1805 \@@_add_to_colors_seq:nn { { nocolor } } { }
1806 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1807 \group_begin:

```

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

```

1808 \if_mode_math:
1809 \@@_exec_code_before_i:
1810 \else:
1811 \c_math_toggle_token
1812 \@@_exec_code_before_i:
1813 \c_math_toggle_token
1814 \fi:
1815 \group_end:
1816 }

```

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`).

```

1817 \cs_new_protected:Npn \@@_exec_code_before_i:
1818 {
1819 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1820 { \@@_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`.

```

1821 \exp_last_unbraced:No \@@_CodeBefore_keys:
1822 \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.

```

1823 \@@_actually_color:
1824 \l_@@_code_before_tl
1825 \q_stop
1826 }

```

```

1827 \keys_define:nn { nicematrix / CodeBefore }
1828 {
1829 create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1830 create-cell-nodes .default:n = true ,
1831 sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1832 sub-matrix .value_required:n = true ,

```

```

1833     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1834     delimiters / color .value_required:n = true ,
1835     unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1836   }

1837   \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1838   {
1839     \keys_set:nn { nicematrix / CodeBefore } { #1 }
1840     \@@_CodeBefore:w
1841   }

```

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.

```

1842   \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1843   {
1844     \bool_if:NT \g_@@_aux_found_bool
1845     {
1846       \@@_pre_code_before:
1847       \legacy_if:nF { measuring@ } { #1 }
1848     }
1849   }

```

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.

```

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

1878   \cs_new_protected:Npn \@@_create_aliases_last:
1879   {
1880     \int_step_inline:nn { \c@iRow }
1881     {
1882       \pgfnodealias
1883       { \@@_env: - ##1 - last }
1884       { \@@_env: - ##1 - \int_use:N \c@jCol }

```

```

1885     }
1886     \int_step_inline:nn { \c@jCol }
1887     {
1888         \pgfnodealias
1889         { \@@_env: - last - ##1 }
1890         { \@@_env: - \int_use:N \c@iRow - ##1 }
1891     }
1892     \pgfnodealias % added 2025-04-05
1893     { \@@_env: - last - last }
1894     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1895 }

```

```

1896 \cs_new_protected:Npn \@@_create_blocks_nodes:
1897 {
1898     \pgfpicture
1899     \pgf@relevantforpicturesizefalse
1900     \pgfrememberpicturepositiononpagetrue
1901     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1902     { \@@_create_one_block_node:nnnnn ##1 }
1903     \endpgfpicture
1904 }

```

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

```

1905 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1906 {
1907     \tl_if_empty:nF { #5 }
1908     {
1909         \@@_qpoint:n { col - #2 }
1910         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1911         \@@_qpoint:n { #1 }
1912         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1913         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1914         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1915         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1916         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1917         \@@_pgf_rect_node:nnnnn
1918         { \@@_env: - #5 }
1919         { \dim_use:N \l_tmpa_dim }
1920         { \dim_use:N \l_tmpb_dim }
1921         { \dim_use:N \l_@@_tmpc_dim }
1922         { \dim_use:N \l_@@_tmpd_dim }
1923     }
1924 }

```

```

1925 \cs_new_protected:Npn \@@_patch_for_revtext:
1926 {
1927     \cs_set_eq:NN \@addamp \@addamp@LaTeX
1928     \cs_set_eq:NN \@array \@array@array
1929     \cs_set_eq:NN \@tabular \@tabular@array
1930     \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1931     \cs_set_eq:NN \array \array@array
1932     \cs_set_eq:NN \endarray \endarray@array
1933     \cs_set:Npn \endtabular { \endarray $\egroup } % $
1934     \cs_set_eq:NN \@mkpream \@mkpream@array
1935     \cs_set_eq:NN \@classx \@classx@array
1936     \cs_set_eq:NN \insert@column \insert@column@array
1937     \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).

```

1938 \cs_set_eq:NN \@xarraycr \@xarraycr@array
1939 \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1940 }

```

10 The environment `{NiceArrayWithDelims}`

```

1941 \NewDocumentEnvironment { NiceArrayWithDelims }
1942 { m m O { } m ! O { } t \CodeBefore }
1943 {
1944 \bool_if:NT \c_@@_revtex_bool { \@@_patch_for_revtex: }
1945 \@@_provide_pgfsyspdfmark:
1946 \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.

```

1947 \bgroup

1948 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1949 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1950 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1951 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

1952 \int_gzero:N \g_@@_block_box_int
1953 \dim_gzero:N \g_@@_width_last_col_dim
1954 \dim_gzero:N \g_@@_width_first_col_dim
1955 \bool_gset_false:N \g_@@_row_of_col_done_bool
1956 \str_if_empty:NT \g_@@_name_env_str
1957 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1958 \bool_if:NTF \l_@@_tabular_bool
1959 { \mode_leave_vertical: }
1960 { \@@_test_if_math_mode: }
1961 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1962 \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.

```

1963 \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.

```

1964 \cs_if_exist:NT \tikz@library@external@loaded
1965 {
1966 \tikzexternaldisable
1967 \cs_if_exist:NT \ifstandalone
1968 { \tikzset { external / optimize = false } }
1969 }

```

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

```

1970 \int_gincr:N \g_@@_env_int
1971 \bool_if:NF \l_@@_block_auto_columns_width_bool
1972 { \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.

```
1973 \seq_gclear:N \g_@@_blocks_seq
1974 \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`.

```
1975 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1976 \seq_gclear:N \g_@@_pos_of_xdots_seq
1977 \tl_gclear_new:N \g_@@_code_before_tl
1978 \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.

```
1979 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1980 {
1981   \bool_gset_true:N \g_@@_aux_found_bool
1982   \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1983 }
1984 { \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.

```
1985 \tl_gclear:N \g_@@_aux_tl
1986 \tl_if_empty:NF \g_@@_code_before_tl
1987 {
1988   \bool_set_true:N \l_@@_code_before_bool
1989   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1990 }
1991 \tl_if_empty:NF \g_@@_pre_code_before_tl
1992 { \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`.

```
1993 \bool_if:NTF \g_@@_delims_bool
1994 { \keys_set:nn { nicematrix / pNiceArray } }
1995 { \keys_set:nn { nicematrix / NiceArray } }
1996 { #3 , #5 }

1997 \@@_set_CTarc:o \l_@@_rules_color_tl % noqa: w302
```

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

```
1998 \bool_if:nTF { #6 } { \@@_CodeBefore_Body:w } { \@@_pre_array: }
1999 }
```

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

```
2000 {
2001   \bool_if:NTF \l_@@_light_syntax_bool
2002   { \use:c { end @@-light-syntax } }
2003   { \use:c { end @@-normal-syntax } }
2004   \c_math_toggle_token
2005   \skip_horizontal:N \l_@@_right_margin_dim
2006   \skip_horizontal:N \l_@@_extra_right_margin_dim
2007   \hbox_set_end:
2008   \bool_if:NT \c_@@_recent_array_bool
2009   { \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.

```

2010 \bool_if:NT \l_@@_width_used_bool
2011 {
2012   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2013   { \@@_error_or_warning:n { width~without~X~columns } }
2014 }

```

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 .

```

2015 \fp_compare:nNnT { \g_@@_total_X_weight_fp } > { \c_zero_fp }
2016 { \@@_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).

```

2017 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
2018 {
2019   \bool_if:NF \l_@@_last_row_without_value_bool
2020   {
2021     \int_compare:nNnF { \l_@@_last_row_int } = { \c@iRow }
2022     {
2023       \@@_error:n { Wrong~last~row }
2024       \int_gset_eq:NN \l_@@_last_row_int \c@iRow
2025     }
2026   }
2027 }

```

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

```

2028 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2029 \bool_if:NTF \g_@@_last_col_found_bool
2030 { \int_gdecr:N \c@jCol }
2031 {
2032   \int_compare:nNnT { \l_@@_last_col_int } > { -1 }
2033   { \@@_error:n { last~col~not~used } }
2034 }

```

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

```

2035 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2036 \int_compare:nNnT { \l_@@_last_row_int } > { -1 }
2037 { \int_gdecr:N \c@iRow }

```

Now, we begin the real construction in the output flow of TeX. First, we take into account a potential “first column” (we remind that this “first column” has been constructed in an overlapping position and that we have computed its width in `\g_@@_width_first_col_dim`: see p. 92).

```

2038 \int_if_zero:nT { \l_@@_first_col_int }
2039 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2040 \bool_if:nTF { ! \g_@@_delims_bool }
2041 {
2042   \str_if_eq:eeTF \l_@@_baseline_tl { c }
2043   { \@@_use_arraybox_with_notes_c: }
2044   {
2045     \str_if_eq:eeTF \l_@@_baseline_tl { b }
2046     { \@@_use_arraybox_with_notes_b: }
2047     { \@@_use_arraybox_with_notes: }
2048   }
2049 }

```

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).

```

2050 {
2051   \int_if_zero:nTF { \l_@@_first_row_int }
2052   {
2053     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2054     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2055   }
2056   { \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

```

2057   \int_compare:nNnTF { \l_@@_last_row_int } > { -2 }
2058   {
2059     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2060     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2061   }
2062   { \dim_zero:N \l_tmpb_dim }
2063   \hbox_set:Nn \l_tmpa_box
2064   {
2065     \m@th
2066     \c_math_toggle_token
2067     \@@_color:o \l_@@_delimiters_color_tl
2068     \exp_after:wN \left \g_@@_left_delim_tl
2069     \vcenter
2070     {

```

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.

```

2071       \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2072       \hbox
2073       {
2074         \bool_if:NTF \l_@@_tabular_bool
2075         { \skip_horizontal:n { - \tabcolsep } }
2076         { \skip_horizontal:n { - \arraycolsep } }
2077         \@@_use_arraybox_with_notes_c:
2078         \bool_if:NTF \l_@@_tabular_bool
2079         { \skip_horizontal:n { - \tabcolsep } }
2080         { \skip_horizontal:n { - \arraycolsep } }
2081       }

```

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

```

2082       \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2083     }
2084     \exp_after:wN \right \g_@@_right_delim_tl
2085     \c_math_toggle_token
2086   }

```

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.

```

2087   \bool_if:NTF \l_@@_delimiters_max_width_bool
2088   {
2089     \@@_put_box_in_flow_bis:nn
2090     { \g_@@_left_delim_tl }
2091     { \g_@@_right_delim_tl }
2092   }
2093   \@@_put_box_in_flow:
2094 }

```

We take into account a potential “last column” (this “last column” has been constructed in an overlapping position and we have computed its width in `\g_@@_width_last_col_dim`: see p. 93).

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).

```

2095 \bool_if:NT \g_@@_last_col_found_bool
2096 { \skip_horizontal:N \g_@@_width_last_col_dim }
2097 \bool_if:NT \l_@@_preamble_bool
2098 {
2099   \int_compare:nNtT { \c@jCol } < { \g_@@_static_num_of_col_int }
2100   { \@@_err_columns_not_used: }
2101 }
2102 \@@_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.

```

2103 \egroup

```

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

```

2104 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2105 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2106 \iow_now:Ne \@mainaux
2107 {
2108   \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2109   \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2110   { \exp_not:o \g_@@_aux_tl }
2111 }
2112 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

```

```

2113 \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2114 }

```

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

```

2115 \cs_new_protected:Npn \@@_err_columns_not_used:
2116 {
2117   \@@_warning:n { columns~not~used }
2118   \cs_gset:Npn \@@_err_columns_not_used: { }
2119 }

```

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 .

```

2120 \cs_new_protected:Npn \@@_compute_width_X:
2121 {
2122   \tl_gput_right:Ne \g_@@_aux_tl
2123   {
2124     \bool_set_true:N \l_@@_X_columns_aux_bool
2125     \dim_set:Nn \l_@@_X_columns_dim
2126     {

```

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

```

2127     \bool_lazy_and:nnTF
2128     { \g_@@_V_of_X_bool }
2129     { \l_@@_X_columns_aux_bool }
2130     { \dim_use:N \l_@@_X_columns_dim }
2131     {
2132       \dim_compare:nNnTF
2133       {
2134         \dim_abs:n
2135         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2136       }
2137       <
2138       { 0.001 pt }
2139       { \dim_use:N \l_@@_X_columns_dim }

```

```

2140         {
2141             \dim_eval:n
2142             {
2143                 \l_@@_X_columns_dim
2144                 +
2145                 \fp_to_dim:n
2146                 {
2147                     (
2148                         \dim_eval:n
2149                         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2150                     )
2151                     / \fp_use:N \g_@@_total_X_weight_fp
2152                 }
2153             }
2154         }
2155     }
2156 }
2157 }
2158 }
2159 }

```

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`.

```

2160 \cs_new_protected:Npn \@@_transform_preamble:
2161 {
2162     \@@_transform_preamble_i:
2163     \@@_transform_preamble_ii:
2164 }
2165 \cs_new_protected:Npn \@@_transform_preamble_i:
2166 {
2167     \int_gzero:N \c@jCol

```

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

```

2168     \seq_gclear:N \g_@@_cols_vlsim_seq

```

`\g_tmpb_bool` will be raised if you have a `|` at the end of the preamble provided by the final user.

```

2169     \bool_gset_false:N \g_tmpb_bool

```

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

```

2170     \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2171     \int_zero:N \l_tmpa_int
2172     \tl_gclear:N \g_@@_array_preamble_tl
2173     \str_if_eq:eeTF \l_@@_vlines_clist { all }
2174     {
2175         \tl_gset:Nn \g_@@_array_preamble_tl
2176         { ! { \skip_horizontal:N \arrayrulewidth } }
2177     }
2178     {
2179         \clist_if_in:NnT \l_@@_vlines_clist 1
2180         {
2181             \tl_gset:Nn \g_@@_array_preamble_tl

```

```

2182         { ! { \skip_horizontal:N \arrayrulewidth } }
2183     }
2184 }

```

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

```

2185     \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2186     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2187     \@@_replace_columncolor:
2188 }

2189 \cs_new_protected:Npn \@@_transform_preamble_ii:
2190 {

```

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

```

2191     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2192     {
2193         \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2194         { \bool_gset_true:N \g_@@_delims_bool }
2195     }
2196     { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2197     \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).

```

2198     \int_if_zero:nTF { \l_@@_first_col_int }
2199     { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2200     {
2201         \bool_if:NF \g_@@_delims_bool
2202         {
2203             \bool_if:NF \l_@@_tabular_bool
2204             {
2205                 \clist_if_empty:NT \l_@@_vlines_clist
2206                 {
2207                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2208                     { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2209                 }
2210             }
2211         }
2212     }
2213     \int_compare:nNnTF { \l_@@_last_col_int } > { -1 }
2214     { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2215     {
2216         \bool_if:NF \g_@@_delims_bool
2217         {
2218             \bool_if:NF \l_@@_tabular_bool
2219             {
2220                 \clist_if_empty:NT \l_@@_vlines_clist
2221                 {
2222                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2223                     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2224                 }
2225             }
2226         }
2227     }

```

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`).

```
2228 \dim_compare:nNnT { \l_@@_tabular_width_dim } = { \c_zero_dim }
2229 {
```

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.

```
2230 \bool_if:NF \c_@@_testphase_table_bool
2231 {
2232   \tl_gput_right:Nn \g_@@_array_preamble_tl
2233   { > { \@@_error_too_much_cols: } 1 }
2234 }
2235 }
2236 }
```

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.

```
2237 \cs_new_protected:Npn \@@_rec_preamble:n #1
2238 {
```

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

```
2239 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2240 { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2241 {
```

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

```
2242 \cs_if_exist:cTF { NC @ find @ #1 }
2243 {
2244   \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2245   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2246 }
2247 {
2248   \str_if_eq:nnTF { #1 } { S }
2249   { \@@_fatal:n { unknown~column~type~S } }
2250   { \@@_fatal:nn { unknown~column~type } { #1 } }
2251 }
2252 }
2253 }
```

For c, l and r

```
2254 \cs_new_protected:Npn \@@_c: #1
2255 {
2256   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2257   \tl_gclear:N \g_@@_pre_cell_tl
2258   \tl_gput_right:Nn \g_@@_array_preamble_tl
2259   { > \@@_cell_begin: c < \@@_cell_end: }
```

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

```
2260 \int_gincr:N \c@jCol
2261 \@@_rec_preamble_after_col:n
2262 }
```

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`.

```

2263 \cs_new_protected:Npn \@@_l: #1
2264 {
2265   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2266   \tl_gclear:N \g_@@_pre_cell_tl
2267   \tl_gput_right:Nn \g_@@_array_preamble_tl
2268   {
2269     > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2270     l
2271     < \@@_cell_end:
2272   }
2273   \int_gincr:N \c@jCol
2274   \@@_rec_preamble_after_col:n
2275 }
2276 \cs_new_protected:Npn \@@_r: #1
2277 {
2278   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2279   \tl_gclear:N \g_@@_pre_cell_tl
2280   \tl_gput_right:Nn \g_@@_array_preamble_tl
2281   {
2282     > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2283     r
2284     < \@@_cell_end:
2285   }
2286   \int_gincr:N \c@jCol
2287   \@@_rec_preamble_after_col:n
2288 }

```

For ! and @

```

2289 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2290 {
2291   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2292   \@@_rec_preamble:n
2293 }
2294 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2295 \cs_new_protected:cpn { @@ _ | : } #1
2296 {
2297   \int_incr:N \l_tmpa_int
2298   \@@_make_preamble_i_i:n
2299 }
2300 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2301 {

```

Here, we can't use \str_if_eq:eeTF.

```

2302   \str_if_eq:nnTF { #1 } { | }
2303   { \use:c { @@ _ | : } | }
2304   { \@@_make_preamble_i_ii:nn { } #1 }
2305 }
2306 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2307 {
2308   \str_if_eq:nnTF { #2 } { [ ] }
2309   { \@@_make_preamble_i_ii:nw { #1 } [ ] }
2310   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2311 }
2312 \cs_new_protected:Npn \@@_make_preamble_i_iii:nw #1 [ #2 ]
2313 { \@@_make_preamble_i_ii:nn { #1 , #2 } }

```

```

2314 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2315 {
2316   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2317   \tl_gput_right:Ne \g_@@_array_preamble_tl
2318   {

```

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

```

2319     \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2320   }
2321   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2322   {
2323     \@@_vline:n
2324     {
2325       position = \int_eval:n { \c@jCol + 1 } ,
2326       multiplicity = \int_use:N \l_tmpa_int ,
2327       total-width = \dim_use:N \l_@@_rule_width_dim ,
2328       #2
2329     }

```

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.

```

2330   }
2331   \int_zero:N \l_tmpa_int
2332   \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2333   \@@_rec_preamble:n #1
2334 }

```

```

2335 \cs_new_protected:cpn { @@_ > : } #1 #2
2336 {
2337   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2338   \@@_rec_preamble:n
2339 }
2340 \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.

```

2341 \keys_define:nn { nicematrix / p-column }
2342 {
2343   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2344   r .value_forbidden:n = true ,
2345   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2346   c .value_forbidden:n = true ,
2347   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2348   l .value_forbidden:n = true ,
2349   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2350   S .value_forbidden:n = true ,
2351   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2352   p .value_forbidden:n = true ,
2353   t .meta:n = p ,
2354   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2355   m .value_forbidden:n = true ,
2356   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2357   b .value_forbidden:n = true
2358 }

```

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

```

2359 \cs_new_protected:Npn \@@_p: #1
2360 {
2361   \str_set:Nn \l_@@_vpos_col_str { #1 }

```


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

```

2362 \@@_make_preamble_ii_i:n
2363 }
2364 \cs_set_eq:NN \@@_b: \@@_p:
2365 \cs_set_eq:NN \@@_m: \@@_p:
2366 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2367 {
2368   \str_if_eq:nnTF { #1 } { [ ] {
2369     { \@@_make_preamble_ii_ii:w [ ] {
2370       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2371     }
2372   }
2373   \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2374   { \@@_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.

```

2374 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2375 {

```

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).

```

2376 \str_set:Nn \l_@@_hpos_col_str { j }
2377 \@@_keys_p_column:n { #1 }

```

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

```

2378 \setlength { \l_tmpa_dim } { #2 }
2379 \@@_make_preamble_ii_iv:nnn { \l_tmpa_dim } { minipage } { }
2380 }
2381 \cs_new_protected:Npn \@@_keys_p_column:n #1
2382 { \keys_set:known { 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.

```

2383 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2384 {
2385   \use:e
2386   {
2387     \@@_make_preamble_ii_vi:nnnnnnnn
2388     { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }
2389     { #1 }
2390     {

```

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.

```

2391 \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2392 { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2393 {

```

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

```

2394 \def \exp_not:N \l_@@_hpos_cell_tl
2395 { \str_lowercase:f { \l_@@_hpos_col_str } }
2396 }
2397 \IfPackageLoadedTF { ragged2e }
2398 {
2399   \str_case:on \l_@@_hpos_col_str
2400   {

```

The following `\exp_not:N` are mandatory.

```

2401         c { \exp_not:N \Centering }
2402         l { \exp_not:N \RaggedRight }
2403         r { \exp_not:N \RaggedLeft }
2404     }
2405 }
2406 {
2407     \str_case:on \l_@@_hpos_col_str
2408     {
2409         c { \exp_not:N \centering }
2410         l { \exp_not:N \raggedright }
2411         r { \exp_not:N \raggedleft }
2412     }
2413 }
2414 #3
2415 }
2416 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2417 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2418 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2419 { #2 }
2420 {
2421     \str_case:onF \l_@@_hpos_col_str
2422     {
2423         { j } { c }
2424         { si } { c }
2425     }

```

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

```

2426         { \str_lowercase:f \l_@@_hpos_col_str }
2427     }
2428 }

```

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

```

2429     \int_gincr:N \c@jCol
2430     \@@_rec_preamble_after_col:n
2431 }

```

#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*.

```

2432 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2433 {
2434     \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2435     {
2436         \tl_gput_right:Nn \g_@@_array_preamble_tl
2437         { > \@@_test_if_empty_for_S: }
2438     }
2439     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2440     \tl_gput_right:Nn \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2441     \tl_gclear:N \g_@@_pre_cell_tl
2442     \tl_gput_right:Nn \g_@@_array_preamble_tl
2443     {
2444         > {

```

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.

```

2445         \dim_set:Nn \l_@@_col_width_dim { #2 }
2446         \bool_if:NT \c_@@_testphase_table_bool
2447           { \tag_struct_begin:n { tag = Div } }
2448         \@@_cell_begin:

```

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

```

2449         \use:c { #7 } [ #1 ] { #2 }

```

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

```

2450         \everypar
2451         {
2452             \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2453             \everypar { }
2454         }
2455         \bool_if:NT \c_@@_testphase_table_bool { \tagpdfpara0n }

```

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

```

2456         #3

```

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

```

2457         \g_@@_row_style_tl
2458         \arraybackslash
2459         #5
2460     }
2461     #8
2462     < {
2463         #6

```

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

```

2464         \@finalstrut \@arstrutbox
2465         \use:c { end #7 }

```

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

```

2466         #4
2467         \@@_cell_end:
2468         \bool_if:NT \c_@@_testphase_table_bool { \tag_struct_end: }
2469     }
2470 }
2471 }

```

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

```

2472 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2473 {

```

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 `&`).

```

2474 \group_align_safe_begin:
2475 \peek_meaning:NTF &
2476 { \@@_the_cell_is_empty: }
2477 {
2478     \peek_meaning:NTF \\\
2479     { \@@_the_cell_is_empty: }
2480     {
2481         \peek_meaning:NTF \crcr
2482         \@@_the_cell_is_empty:
2483         \group_align_safe_end:
2484     }
2485 }
2486 }

```

```

2487 \cs_new_protected:Npn \@@_the_cell_is_empty:
2488 {
2489   \group_align_safe_end:
2490   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2491   {

```

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

```

2492     \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2493     \skip_horizontal:N \l_@@_col_width_dim
2494   }
2495 }
2496 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2497 {
2498   \peek_meaning:NT \__siunitx_table_skip:n
2499   { \bool_gset_true:N \g_@@_empty_cell_bool }
2500 }

```

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.

```

2501 \cs_new_protected:Npn \@@_center_cell_box:
2502 {

```

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

```

2503   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2504   {
2505     \dim_compare:nNnT
2506     { \box_ht:N \l_@@_cell_box }
2507     >

```

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).

```

2508     { \box_ht:N \strutbox }
2509     {
2510       \hbox_set:Nn \l_@@_cell_box
2511       {
2512         \box_move_down:nn
2513         {
2514           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2515             + \baselineskip ) / 2
2516         }
2517         { \box_use:N \l_@@_cell_box }
2518       }
2519     }
2520   }
2521 }

```

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

```

2522 \cs_new_protected:Npn \@@_V: #1 #2
2523 {
2524   \str_if_eq:nnTF { #1 } { [ ] }
2525   { \@@_make_preamble_V_i:w [ ] }
2526   { \@@_make_preamble_V_i:w [ ] { #2 } }
2527 }
2528 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2529 { \@@_make_preamble_V_ii:nn { #1 } }
2530 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2

```

```

2531 {
2532   \str_set:Nn \l_@@_vpos_col_str { p }
2533   \str_set:Nn \l_@@_hpos_col_str { j }
2534   \@@_keys_p_column:n { #1 }

```

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

```

2535   \setlength { \l_tmpa_dim } { #2 }
2536   \IfPackageLoadedTF { varwidth }
2537     { \@@_make_preamble_ii_iv:nnn { \l_tmpa_dim } { varwidth } { } }
2538     {
2539       \@@_error_or_warning:n { varwidth-not-loaded }
2540       \@@_make_preamble_ii_iv:nnn { \l_tmpa_dim } { minipage } { }
2541     }
2542 }

```

For `w` and `W`

```

2543 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2544 \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.

```

2545 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2546 {
2547   \str_if_eq:nnTF { #3 } { s }
2548     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2549     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2550 }

```

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.

```

2551 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2552 {
2553   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2554   \tl_gclear:N \g_@@_pre_cell_tl
2555   \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2556   {
2557     > {
2558       % We use |\setlength| in order to allow |\widthof| which is a command of \pkg{calc}
2559       % (when loaded \pkg{calc} redefines |\setlength|).
2560       % Of course, even if \pkg{calc} is not loaded, the following code will work with
2561       % the standard version of |\setlength|.
2562       \setlength { \l_@@_col_width_dim } { #2 }
2563       \@@_cell_begin:
2564       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2565     }
2566     c
2567     < {
2568       \@@_cell_end_for_w_s:
2569       #1
2570       \@@_adjust_size_box:
2571       \box_use_drop:N \l_@@_cell_box
2572     }
2573   }
2574   \int_gincr:N \c@jCol
2575   \@@_rec_preamble_after_col:n
2576 }

```

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

```

2577 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2578 {
2579   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2580   \tl_gclear:N \g_@@_pre_cell_tl
2581   \tl_gput_right:Nn \g_@@_array_preamble_tl
2582   {
2583     > {

```

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

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

```

2584       \setlength { \l_@@_col_width_dim } { #4 }
2585       \hbox_set:Nw \l_@@_cell_box
2586       \@@_cell_begin:
2587       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2588     }
2589     c
2590     < {
2591       \@@_cell_end:
2592       \hbox_set_end:
2593       #1
2594       \@@_adjust_size_box:
2595       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2596     }
2597   }

```

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

```

2598   \int_gincr:N \c@jCol
2599   \@@_rec_preamble_after_col:n
2600 }

```

```

2601 \cs_new_protected:Npn \@@_special_W:
2602 {
2603   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \l_@@_col_width_dim }
2604   { \@@_warning:n { W~warning } }
2605 }

```

For `S` (of `siunitx`).

```

2606 \cs_new_protected:Npn \@@_S: #1 #2
2607 {
2608   \str_if_eq:nnTF { #2 } { [ ] }
2609   { \@@_make_preamble_S:w [ ] }
2610   { \@@_make_preamble_S:w [ ] { #2 } }
2611 }
2612 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2613 { \@@_make_preamble_S_i:n { #1 } }
2614 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2615 {
2616   \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2617   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2618   \tl_gclear:N \g_@@_pre_cell_tl
2619   \tl_gput_right:Nn \g_@@_array_preamble_tl
2620   {
2621     > {

```

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).

```

2622         \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2623         \keys_set:nn { siunitx } { #1 }
2624         \@@_cell_begin:
2625         \siunitx_cell_begin:w
2626     }
2627     c
2628     <
2629     {
2630         \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, it will stay local within the cell of the underlying `\halign`).

```

2631         \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2632         {
2633             \bool_if:NTF \l__siunitx_table_text_bool
2634             { \bool_set_true:N }
2635             { \bool_set_false:N }
2636             \l__siunitx_table_text_bool
2637         }
2638         \@@_cell_end:
2639     }
2640 }

```

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

```

2641     \int_gincr:N \c@jCol
2642     \@@_rec_preamble_after_col:n
2643 }

```

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

```

2644 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2645 {
2646     \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.

```

2647     \int_if_zero:nTF { \c@jCol }
2648     {
2649         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2650         {

```

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

```

2651         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2652         \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2653         \@@_rec_preamble:n #2
2654     }
2655     {
2656         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2657         \@@_make_preamble_iv:nn { #1 } { #2 }
2658     }
2659 }
2660 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2661 }
2662 \cs_set_eq:cc { @@ _ \token_to_str:N [ : } { @@ _ \token_to_str:N ( : }
2663 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2664 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2665 {
2666     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2667     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2668     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2669     {
2670         \@@_error:nn { delimiter~after~opening } { #2 }
2671         \@@_rec_preamble:n

```

```

2672     }
2673     { \@@_rec_preamble:n #2 }
2674 }

```

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

```

2675 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2676 { \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}`).

```

2677 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2678 {
2679   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2680   \tl_if_in:nnTF { ) } [ \ ] { #2 }
2681   { \@@_make_preamble_v:nnn #1 #2 }
2682   {
2683     \str_if_eq:nnTF { \s_stop } { #2 }
2684     {
2685       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2686       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2687       {
2688         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2689         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2690         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2691         \@@_rec_preamble:n #2
2692       }
2693     }
2694     {
2695       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2696       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2697       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2698       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2699       \@@_rec_preamble:n #2
2700     }
2701   }
2702 }
2703 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2704 \cs_set_eq:cc { @@ _ \token_to_str:N \ } : } { @@ _ \token_to_str:N ) : }
2705 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2706 {
2707   \str_if_eq:nnTF { \s_stop } { #3 }
2708   {
2709     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2710     {
2711       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2712       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2713       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2714       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2715     }
2716     {
2717       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2718       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2719       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2720       \@@_error:nn { double~closing~delimiter } { #2 }
2721     }
2722   }
2723   {
2724     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2725     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2726     \@@_error:nn { double~closing~delimiter } { #2 }
2727     \@@_rec_preamble:n #3

```



```

2728     }
2729 }

2730 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2731 { \use:c { @@ _ \token_to_str:N ) : } }

```

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

```

2732 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2733 {
2734   \str_if_eq:nnTF { #1 } { < }
2735   { \@@_rec_preamble_after_col_i:n }
2736   {
2737     \str_if_eq:nnTF { #1 } { @ }
2738     { \@@_rec_preamble_after_col_ii:n }
2739     {
2740       \str_if_eq:eeTF \l_@@_vlines_clist { all }
2741       {
2742         \tl_gput_right:Nn \g_@@_array_preamble_tl
2743         { ! { \skip_horizontal:N \arrayrulewidth } }
2744       }
2745       {
2746         \clist_if_in:NeT \l_@@_vlines_clist
2747         { \int_eval:n { \c@jCol + 1 } }
2748         {
2749           \tl_gput_right:Nn \g_@@_array_preamble_tl
2750           { ! { \skip_horizontal:N \arrayrulewidth } }
2751         }
2752       }
2753       \@@_rec_preamble:n { #1 }
2754     }
2755   }
2756 }

2757 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2758 {
2759   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2760   \@@_rec_preamble_after_col:n
2761 }

```

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

```

2762 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2763 {
2764   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2765   {
2766     \tl_gput_right:Nn \g_@@_array_preamble_tl
2767     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2768   }
2769   {
2770     \clist_if_in:NeTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2771     {
2772       \tl_gput_right:Nn \g_@@_array_preamble_tl
2773       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2774     }
2775     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2776   }
2777   \@@_rec_preamble:n
2778 }

2779 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3

```

```

2780 {
2781   \tl_clear:N \l_tmpa_tl
2782   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2783   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2784 }

```

The token `\NC@find` is at the head of the definition of the columns type done by `\newcolumntype`. We want that token to be no-op here.

```

2785 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2786 { \@@_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.

```

2787 \cs_new_protected:Npn \@@_X: #1 #2
2788 {
2789   \str_if_eq:nnTF { #2 } { [ ]
2790     { \@@_make_preamble_X:w [ ] }
2791     { \@@_make_preamble_X:w [ ] #2 }
2792   }
2793   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2794   { \@@_make_preamble_X_i:n { #1 } }

```

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

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

```

2795 \keys_define:nn { nicematrix / X-column }
2796 {
2797   V .code:n =
2798     \IfPackageLoadedTF { varwidth }
2799     {
2800       \bool_set_true:N \l_@@_V_of_X_bool
2801       \bool_gset_true:N \g_@@_V_of_X_bool
2802     }
2803     { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
2804   unknown .code:n =
2805     \regex_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2806     { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2807     { \@@_error_or_warning:n { invalid~weight } }
2808 }

```

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

```

2809 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2810 {

```

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).

```

2811   \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).

```

2812   \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`).

```

2813   \fp_set:Nn \l_tmpa_fp { 1.0 }
2814   \@@_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.

```
2815 \bool_set_false:N \l_@@_V_of_X_bool
2816 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
```

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

```
2817 \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).

```
2818 \bool_if:NTF \l_@@_X_columns_aux_bool
2819 {
2820 \@@_make_preamble_ii_iv:nnn
```

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

```
2821 { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2822 { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
2823 { \@@_no_update_width: }
2824 }
```

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.

```
2825 {
2826 \tl_gput_right:Nn \g_@@_array_preamble_tl
2827 {
2828 > {
2829 \@@_cell_begin:
2830 \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`.

```
2831 \NotEmpty
```

The following code will nullify the box of the cell.

```
2832 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2833 { \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`.

```
2834 \begin { minipage } { 5 cm } \arraybackslash
2835 }
2836 c
2837 < {
2838 \end { minipage }
2839 \@@_cell_end:
2840 }
2841 }
2842 \int_gincr:N \c@jCol
2843 \@@_rec_preamble_after_col:n
2844 }
2845 }
```

```
2846 \cs_new_protected:Npn \@@_no_update_width:
2847 {
2848 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2849 { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2850 }
```

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

```

2851 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2852 {
2853   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2854   { \int_eval:n { \c@jCol + 1 } }
2855   \tl_gput_right:Ne \g_@@_array_preamble_tl
2856   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2857   \@@_rec_preamble:n
2858 }

```

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.

```

2859 \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).

```

2860 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2861 { \@@_fatal:n { Preamble-forgotten } }
2862 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
2863 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
2864 { @@ _ \token_to_str:N \hline : }
2865 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
2866 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
2867 { @@ _ \token_to_str:N \hline : }
2868 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
2869 { @@ _ \token_to_str:N \hline : }
2870 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
2871 { @@ _ \token_to_str:N \hline : }
2872 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
2873 { @@ _ \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).

```

2874 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2875 {

```

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`.

```

2876   \multispan { #1 }
2877   \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2878   \begingroup
2879   \bool_if:NT \c_@@_testphase_table_bool
2880   { \tbl_update_multicolumn_cell_data:n { #1 } }
2881   \def \@@addamp
2882   { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2883   \tl_gclear:N \g_@@_preamble_tl
2884   \@@_make_m_preamble:n #2 \q_stop

```

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

```

2885   \exp_args:No \@mkpream \g_@@_preamble_tl
2886   \@addtopreamble \@empty
2887   \endgroup
2888   \bool_if:NT \c_@@_recent_array_bool
2889   { \UseTaggingSocket { tbl / colspan } { #1 } }

```

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

```

2890 \int_compare:nNtT { #1 } > { \c_one_int }
2891 {
2892   \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2893   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2894   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2895   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2896   {
2897     {
2898       \int_if_zero:nTF { \c@jCol }
2899       { \int_eval:n { \c@iRow + 1 } }
2900       { \int_use:N \c@iRow }
2901     }
2902     { \int_eval:n { \c@jCol + 1 } }
2903     {
2904       \int_if_zero:nTF { \c@jCol }
2905       { \int_eval:n { \c@iRow + 1 } }
2906       { \int_use:N \c@iRow }
2907     }
2908     { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block

```

2909     { }
2910   }
2911 }

```

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

```

2912 \RenewDocumentCommand { \cellcolor } { 0 { } m }
2913 {
2914   \tl_gput_right:Ne \g_@@_pre_code_before_tl
2915   {
2916     \@@_rectanglecolor [ ##1 ]
2917     { \exp_not:n { ##2 } }
2918     { \int_use:N \c@iRow - \int_use:N \c@jCol }
2919     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2920   }
2921   \ignorespaces
2922 }

```

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

```

2923 \def \@sharp { #3 }
2924 \@arstrut
2925 \@preamble
2926 \null

```

We add some lines.

```

2927 \int_gadd:Nn \c@jCol { #1 - 1 }
2928 \int_compare:nNtT { \c@jCol } > { \g_@@_col_total_int }
2929 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2930 \ignorespaces
2931 }

```

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`.

```

2932 \cs_new_protected:Npn \@_make_m_preamble:n #1
2933 {
2934   \str_case:nnF { #1 }
2935   {
2936     c { \@_make_m_preamble_i:n #1 }
2937     l { \@_make_m_preamble_i:n #1 }

```

```

2938 r { \@@_make_m_preamble_i:n #1 }
2939 > { \@@_make_m_preamble_ii:nn #1 }
2940 ! { \@@_make_m_preamble_ii:nn #1 }
2941 @ { \@@_make_m_preamble_ii:nn #1 }
2942 | { \@@_make_m_preamble_iii:n #1 }
2943 p { \@@_make_m_preamble_iv:nnn t #1 }
2944 m { \@@_make_m_preamble_iv:nnn c #1 }
2945 b { \@@_make_m_preamble_iv:nnn b #1 }
2946 w { \@@_make_m_preamble_v:nnnn { } #1 }
2947 W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2948 \q_stop { }
2949 }
2950 {
2951 \cs_if_exist:cTF { NC @ find @ #1 }
2952 {
2953 \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2954 \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2955 }
2956 {
2957 \str_if_eq:nnTF { #1 } { S }
2958 { \@@_fatal:n { unknown~column~type~S~multicolumn } }
2959 { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
2960 }
2961 }
2962 }

```

For c, l and r

```

2963 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2964 {
2965 \tl_gput_right:Nn \g_@@_preamble_tl
2966 {
2967 > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
2968 #1
2969 < \@@_cell_end:
2970 }

```

We test for the presence of a <.

```

2971 \@@_make_m_preamble_x:n
2972 }

```

For >, ! and @

```

2973 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2974 {
2975 \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2976 \@@_make_m_preamble:n
2977 }

```

For |

```

2978 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2979 {
2980 \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2981 \@@_make_m_preamble:n
2982 }

```

For p, m and b

```

2983 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2984 {
2985 \tl_gput_right:Nn \g_@@_preamble_tl
2986 {
2987 > {
2988 \@@_cell_begin:

```

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

```

2989         \setlength { \l_tmpa_dim } { #3 }
2990         \begin { minipage } [ #1 ] { \l_tmpa_dim }
2991         \mode_leave_vertical:
2992         \arraybackslash
2993         \vrule height \box_ht:N \@arstrutbox depth \c_zero_dim width \c_zero_dim
2994     }
2995     c
2996     < {
2997         \vrule height \c_zero_dim depth \box_dp:N \@arstrutbox width \c_zero_dim
2998         \end { minipage }
2999         \@@_cell_end:
3000     }
3001 }
```

We test for the presence of a `<`.

```

3002     \@@_make_m_preamble_x:n
3003 }
```

For `w` and `W`

```

3004 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3005 {
3006     \tl_gput_right:Nn \g_@@_preamble_tl
3007     {
3008         > {
3009             \dim_set:Nn \l_@@_col_width_dim { #4 }
3010             \hbox_set:Nw \l_@@_cell_box
3011             \@@_cell_begin:
3012             \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3013         }
3014         c
3015         < {
3016             \@@_cell_end:
3017             \hbox_set_end:
3018             \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3019             #1
3020             \@@_adjust_size_box:
3021             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3022         }
3023     }
}
```

We test for the presence of a `<`.

```

3024     \@@_make_m_preamble_x:n
3025 }
```

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

```

3026 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3027 {
3028     \str_if_eq:nnTF { #1 } { < }
3029     { \@@_make_m_preamble_ix:n }
3030     { \@@_make_m_preamble:n { #1 } }
3031 }
3032 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3033 {
3034     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3035     \@@_make_m_preamble_x:n
3036 }
```

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`).

```

3037 \cs_new_protected:Npn \@@_put_box_in_flow:
3038 {
3039   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3040   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3041   \str_if_eq:eeTF \l_@@_baseline_tl { c }
3042     { \box_use_drop:N \l_tmpa_box }
3043     { \@@_put_box_in_flow_i: }
3044 }

```

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

```

3045 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3046 {
3047   \pgfpicture
3048     \@@_qpoint:n { row - 1 }
3049     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3050     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3051     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3052     \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).

```

3053   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3054   {
3055     \int_set:Nn \l_tmpa_int
3056     {
3057       \str_range:Nnn
3058         \l_@@_baseline_tl
3059         6
3060         { \tl_count:o \l_@@_baseline_tl }
3061     }
3062     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3063   }
3064   {
3065     \str_if_eq:eeTF \l_@@_baseline_tl { t }
3066     { \int_set_eq:NN \l_tmpa_int \c_one_int }
3067     {
3068       \str_if_eq:onTF \l_@@_baseline_tl { b }
3069       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3070       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3071     }
3072     \bool_lazy_or:nnT
3073     { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3074     { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3075     {
3076       \@@_error:n { bad-value~for~baseline }
3077       \int_set_eq:NN \l_tmpa_int \c_one_int
3078     }
3079     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3080     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3081   }
3082   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3083   \endpgfpicture
3084   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3085   \box_use_drop:N \l_tmpa_box
3086 }

```


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).

```
3087 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3088 {
```

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.

```
3089 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3090 {
3091   \int_compare:nNt { \c@jCol } > { \c_one_int }
3092   {
3093     \box_set_wd:Nn \l_@@_the_array_box
3094     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3095   }
3096 }
```

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

```
3097 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3098 \bool_if:NT \l_@@_caption_above_bool
3099 {
3100   \tl_if_empty:NF \l_@@_caption_tl
3101   {
3102     \bool_set_false:N \g_@@_caption_finished_bool
3103     \int_gzero:N \c@tabularnote
3104     \@@_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).

```
3105     \int_compare:nNt { \g_@@_notes_caption_int } > { \c_zero_int }
3106     {
3107       \tl_gput_right:Ne \g_@@_aux_tl
3108       {
3109         \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3110         { \int_use:N \g_@@_notes_caption_int }
3111       }
3112       \int_gzero:N \g_@@_notes_caption_int
3113     }
3114   }
3115 }
```

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

```
3116 \hbox
3117 {
3118   \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.

```
3119 \@@_create_extra_nodes:
3120 \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3121 }
```

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).

```
3122 \bool_lazy_any:nT
3123 {
3124   { ! \seq_if_empty_p:N \g_@@_notes_seq }
```

```

3125     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3126     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3127   }
3128   \@@_insert_tabularnotes:
3129   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3130   \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3131   \end { minipage }
3132 }

```

```

3133 \cs_new_protected:Npn \@@_insert_caption:
3134 {
3135   \tl_if_empty:NF \l_@@_caption_tl
3136   {
3137     \cs_if_exist:NTF \@capytype
3138     { \@@_insert_caption_i: }
3139     { \@@_error:n { caption~outside~float } }
3140   }
3141 }

```

```

3142 \cs_new_protected:Npn \@@_insert_caption_i:
3143 {
3144   \group_begin:

```

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

```

3145   \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.

```

3146   \IfPackageLoadedT { floatrow }
3147   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3148   \tl_if_empty:NTF \l_@@_short_caption_tl
3149   { \caption }
3150   { \caption [ \l_@@_short_caption_tl ] }
3151   { \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.

```

3152   \bool_if:NF \g_@@_caption_finished_bool
3153   {
3154     \bool_gset_true:N \g_@@_caption_finished_bool
3155     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3156     \int_gzero:N \c@tabularnote
3157   }
3158   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3159   \group_end:
3160 }

```

```

3161 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3162 {
3163   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3164   \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3165 }

```

```

3166 \cs_new_protected:Npn \@@_insert_tabularnotes:
3167 {
3168   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3169   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3170   \skip_vertical:N 0.65ex

```

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

```

3171 \group_begin:
3172 \l_@@_notes_code_before_tl
3173 \tl_if_empty:NF \g_@@_tabularnote_tl
3174 {
3175   \g_@@_tabularnote_tl \par
3176   \tl_gclear:N \g_@@_tabularnote_tl
3177 }

```

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.

```

3178 \int_compare:nNtT { \c@tabularnote } > { \c_zero_int }
3179 {
3180   \bool_if:NTF \l_@@_notes_para_bool
3181   {
3182     \begin { tabularnotes* }
3183     \seq_map_inline:Nn \g_@@_notes_seq
3184       { \@@_one_tabularnote:nn ##1 }
3185     \strut
3186     \end { tabularnotes* }

```

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

```

3187 \par
3188 }
3189 {
3190   \tabularnotes
3191   \seq_map_inline:Nn \g_@@_notes_seq
3192     { \@@_one_tabularnote:nn ##1 }
3193   \strut
3194   \endtabularnotes
3195 }
3196 }
3197 \unskip
3198 \group_end:
3199 \bool_if:NT \l_@@_notes_bottomrule_bool
3200 {
3201   \IfPackageLoadedTF { booktabs }
3202   {

```

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

```

3203 \skip_vertical:N \aboverulesep

```

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

```

3204 { \CT@arc@ \hrule height \heavyrulewidth }
3205 }
3206 { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3207 }
3208 \l_@@_notes_code_after_tl
3209 \seq_gclear:N \g_@@_notes_seq
3210 \seq_gclear:N \g_@@_notes_in_caption_seq
3211 \int_gzero:N \c@tabularnote
3212 }

```

The following command will format (after the main tabular) one tabularnote (with the command `\item`). `#1` is the label (when the command `\tabularnote` has been used with an optional argument between square brackets) and `#2` is the text of the note. The second argument is provided by currying.

```

3213 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3214 {
3215   \tl_if_novalue:nTF { #1 }
3216   { \item }
3217   { \item [ \@@_notes_label_in_list:n { #1 } ] }
3218 }

```

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`.

```

3219 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3220 {
3221   \pgfpicture
3222     \@@_qpoint:n { row - 1 }
3223     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3224     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3225     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3226   \endpgfpicture
3227   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3228   \int_if_zero:nT { \l_@@_first_row_int }
3229   {
3230     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3231     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3232   }
3233   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3234 }

```

Now, the general case.

```

3235 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3236 {

```

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

```

3237   \str_if_eq:eeT \l_@@_baseline_tl { t }
3238   { \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`.

```

3239   \pgfpicture
3240   \@@_qpoint:n { row - 1 }
3241   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3242   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3243   {
3244     \int_set:Nn \l_tmpa_int
3245     {
3246       \str_range:Nnn
3247         \l_@@_baseline_tl
3248         { 6 }
3249         { \tl_count:o \l_@@_baseline_tl }
3250     }
3251     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3252   }
3253   {
3254     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3255     \bool_lazy_or:nnT
3256       { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3257       { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3258     {
3259       \@@_error:n { bad-value-for-baseline }
3260       \int_set:Nn \l_tmpa_int 1
3261     }
3262     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3263   }
3264   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3265   \endpgfpicture
3266   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3267   \int_if_zero:nT { \l_@@_first_row_int }
3268   {
3269     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3270     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3271   }
3272   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }

```

```
3273 }
```

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.

```
3274 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3275 {
```

We will compute the real width of both delimiters used.

```
3276 \dim_zero_new:N \l_@@_real_left_delim_dim
3277 \dim_zero_new:N \l_@@_real_right_delim_dim
3278 \hbox_set:Nn \l_tmpb_box
3279 {
3280 \m@th % added 2024/11/21
3281 \c_math_toggle_token
3282 \left #1
3283 \vcenter
3284 {
3285 \vbox_to_ht:nn
3286 { \box_ht_plus_dp:N \l_tmpa_box }
3287 { }
3288 }
3289 \right .
3290 \c_math_toggle_token
3291 }
3292 \dim_set:Nn \l_@@_real_left_delim_dim
3293 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3294 \hbox_set:Nn \l_tmpb_box
3295 {
3296 \m@th % added 2024/11/21
3297 \c_math_toggle_token
3298 \left .
3299 \vbox_to_ht:nn
3300 { \box_ht_plus_dp:N \l_tmpa_box }
3301 { }
3302 \right #2
3303 \c_math_toggle_token
3304 }
3305 \dim_set:Nn \l_@@_real_right_delim_dim
3306 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

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

```
3307 \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3308 \@@_put_box_in_flow:
3309 \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3310 }
```

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).

```
3311 \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).

```
3312 {
3313 \peek_remove_spaces:n
3314 {
3315 \peek_meaning:NTF \end
3316 { \@@_analyze_end:Nn }
```

```

3317     {
3318         \@@_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`).

```

3319         \@@_array:o \g_@@_array_preamble_tl
3320     }
3321 }
3322 }
3323 {
3324     \@@_create_col_nodes:
3325     \endarray
3326 }

```

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

```

3327 \NewDocumentEnvironment { @@-light-syntax } { b }
3328 {

```

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`.

```

3329     \tl_if_empty:nT { #1 }
3330     { \@@_fatal:n { empty-environment } }
3331     \tl_if_in:nnT { #1 } { & }
3332     { \@@_fatal:n { ampersand-in~light-syntax } }
3333     \tl_if_in:nnT { #1 } { \ }
3334     { \@@_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`.

```

3335     \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3336 }

```

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.

```

3337 {
3338     \@@_create_col_nodes:
3339     \endarray
3340 }
3341 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3342 {
3343     \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).

```

3344     \seq_clear_new:N \l_@@_rows_seq

```

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

```

3345     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3346     \bool_if:NTF \l_@@_light_syntax_expanded_bool
3347     { \seq_set_split:Nee }
3348     { \seq_set_split:Non }
3349     \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3350     \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3351     \tl_if_empty:NF \l_tmpa_tl
3352     { \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.

```

3353   \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
3354   { \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`).

```

3355   \tl_build_begin:N \l_@@_new_body_tl
3356   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3357   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3358   \@@_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).

```

3359   \seq_map_inline:Nn \l_@@_rows_seq
3360   {
3361     \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3362     \@@_line_with_light_syntax:n { #1 }
3363   }
3364   \tl_build_end:N \l_@@_new_body_tl
3365   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
3366   {
3367     \int_set:Nn \l_@@_last_col_int
3368     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3369   }

```

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.

```

3370   \@@_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`).

```

3371   \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3372   }
3373   \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3374   {
3375     \seq_clear_new:N \l_@@_cells_seq
3376     \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3377     \int_set:Nn \l_@@_nb_cols_int
3378     {
3379       \int_max:nn
3380       { \l_@@_nb_cols_int }
3381       { \seq_count:N \l_@@_cells_seq }
3382     }
3383     \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3384     \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3385     \seq_map_inline:Nn \l_@@_cells_seq
3386     { \tl_build_put_right:Nn \l_@@_new_body_tl { & #1 } }
3387   }
3388   \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`.

```

3389   \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3390   {
3391     \str_if_eq:eeT \g_@@_name_env_str { #2 }
3392     { \@@_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.

```
3393   \end { #2 }
3394 }
```

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`).

```
3395 \cs_new:Npn \@@_create_col_nodes:
3396 {
3397   \crr
3398   \int_if_zero:nT { \l_@@_first_col_int }
3399   {
3400     \omit
3401     \hbox_overlap_left:n
3402     {
3403       \bool_if:NT \l_@@_code_before_bool
3404       { \pgfsys@markposition { \@@_env: - col - 0 } }
3405       \pgfpicture
3406       \pgfrememberpicturepositiononpagetrue
3407       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3408       \str_if_empty:NF \l_@@_name_str
3409       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3410       \endpgfpicture
3411       \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3412     }
3413     &
3414   }
3415   \omit
```

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

```
3416   \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`).

```
3417   \int_if_zero:nTF { \l_@@_first_col_int }
3418   {
3419     \@@_mark_position:n { 1 }
3420     \pgfpicture
3421     \pgfrememberpicturepositiononpagetrue
3422     \pgfcoordinate { \@@_env: - col - 1 }
3423     { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
3424     \str_if_empty:NF \l_@@_name_str
3425     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3426     \endpgfpicture
3427   }
3428   {
3429     \bool_if:NT \l_@@_code_before_bool
3430     {
3431       \hbox
3432       {
3433         \skip_horizontal:n { 0.5 \arrayrulewidth }
3434         \pgfsys@markposition { \@@_env: - col - 1 }
3435         \skip_horizontal:n { -0.5 \arrayrulewidth }
3436       }
3437     }
3438     \pgfpicture
3439     \pgfrememberpicturepositiononpagetrue
3440     \pgfcoordinate { \@@_env: - col - 1 }
3441     { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }
3442     \@@_node_alias:n { 1 }
3443     \endpgfpicture
3444   }
```


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.

```

3445 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3446 \bool_if:NF \l_@@_auto_columns_width_bool
3447 { \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim } }
3448 {
3449   \bool_lazy_and:nnTF
3450     { \l_@@_auto_columns_width_bool }
3451     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3452     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3453     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3454   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3455 }
3456 \skip_horizontal:N \g_tmpa_skip
3457 \hbox
3458 {
3459   \@@_mark_position:n { 2 }
3460   \pgfpicture
3461   \pgfrememberpicturepositiononpagetrue
3462   \pgfcoordinate { \@@_env: - col - 2 }
3463     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3464   \@@_node_alias:n { 2 }
3465   \endpgfpicture
3466 }

```

We begin a loop over the columns. The integer `\g_tmpa_int` will be the number of the current column. This integer is used for the Tikz nodes.

```

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

```

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

```

3475 \skip_horizontal:N \g_tmpa_skip
3476 \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

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

```

3477 \pgfpicture
3478 \pgfrememberpicturepositiononpagetrue
3479 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3480   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3481   \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3482 \endpgfpicture
3483 }

3484 &
3485 \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).

```

3486 \bool_lazy_or:nnF
3487 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3488 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3489 {
3490   \skip_horizontal:N \g_tmpa_skip

```

```

3491 \int_gincr:N \g_tmpa_int
3492 \bool_lazy_any:nF
3493 {
3494   \g_@@_delims_bool
3495   \l_@@_tabular_bool
3496   { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3497   \l_@@_exterior_arraycolsep_bool
3498   \l_@@_bar_at_end_of_pream_bool
3499 }
3500 { \skip_horizontal:n { - \col@sep } }
3501 \bool_if:NT \l_@@_code_before_bool
3502 {
3503   \hbox
3504   {
3505     \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.

```

3506   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3507   { \skip_horizontal:n { - \arraycolsep } }
3508   \pgfsys@markposition
3509   { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3510   \skip_horizontal:n { 0.5 \arrayrulewidth }
3511   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3512   { \skip_horizontal:N \arraycolsep }
3513 }
3514 }
3515 \pgfpicture
3516 \pgfrememberpicturepositiononpagetrue
3517 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3518 {
3519   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3520   {
3521     \pgfpoint
3522     { - 0.5 \arrayrulewidth - \arraycolsep }
3523     \c_zero_dim
3524   }
3525   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3526 }
3527 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3528 \endpgfpicture
3529 }

```

```

3530 \bool_if:NT \g_@@_last_col_found_bool
3531 {
3532   \hbox_overlap_right:n
3533   {
3534     \skip_horizontal:N \g_@@_width_last_col_dim
3535     \skip_horizontal:N \col@sep
3536     \bool_if:NT \l_@@_code_before_bool
3537     {
3538       \pgfsys@markposition
3539       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3540     }
3541     \pgfpicture
3542     \pgfrememberpicturepositiononpagetrue
3543     \pgfcoordinate
3544     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3545     \pgfpointorigin
3546     \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3547     \endpgfpicture
3548   }

```

```

3549     }
3550     % \cr
3551 }

3552 \cs_new_protected:Npn \@@_mark_position:n #1
3553 {
3554     \bool_if:NT \l_@@_code_before_bool
3555     {
3556         \hbox
3557         {
3558             \skip_horizontal:n { -0.5 \arrayrulewidth }
3559             \pgfsys@markposition { \@@_env: - col - #1 }
3560             \skip_horizontal:n { 0.5 \arrayrulewidth }
3561         }
3562     }
3563 }

3564 \cs_new_protected:Npn \@@_node_alias:n #1
3565 {
3566     \str_if_empty:NF \l_@@_name_str
3567     { \pgfnodealias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3568 }

```

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

```

3569 \tl_const:Nn \c_@@_preamble_first_col_tl
3570 {
3571     >
3572     {

```

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

```

3573         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3574         \bool_gset_true:N \g_@@_after_col_zero_bool
3575         \@@_begin_of_row:
3576         \hbox_set:Nw \l_@@_cell_box
3577         \@@_math_toggle:
3578         \@@_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”.

```

3579         \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3580         {
3581             \bool_lazy_or:nnT
3582             { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3583             { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3584             {
3585                 \l_@@_code_for_first_col_tl
3586                 \xglobal \colorlet { nicematrix-first-col } { . }
3587             }
3588         }
3589     }

```

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`.

```

3590     l
3591     <
3592     {
3593         \@@_math_toggle:
3594         \hbox_set_end:
3595         \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3596         \@@_adjust_size_box:
3597         \@@_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.

```

3598     \dim_gset:Nn \g_@@_width_first_col_dim
3599     { \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.

```

3600     \hbox_overlap_left:n
3601     {
3602         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3603         { \@@_node_cell: }
3604         { \box_use_drop:N \l_@@_cell_box }
3605         \skip_horizontal:N \l_@@_left_delim_dim
3606         \skip_horizontal:N \l_@@_left_margin_dim
3607         \skip_horizontal:N \l_@@_extra_left_margin_dim
3608     }
3609     \bool_gset_false:N \g_@@_empty_cell_bool
3610     \skip_horizontal:n { -2 \col@sep }
3611 }
3612 }

```

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

```

3613 \tl_const:Nn \c_@@_preamble_last_col_tl
3614 {
3615     >
3616     {
3617         \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).

```

3618         \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.

```

3619         \bool_gset_true:N \g_@@_last_col_found_bool
3620         \int_gincr:N \c@jCol
3621         \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3622         \hbox_set:Nw \l_@@_cell_box
3623         \@@_math_toggle:
3624         \@@_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”.

```

3625         \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3626         {
3627             \bool_lazy_or:nnT
3628             { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3629             { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3630             {
3631                 \l_@@_code_for_last_col_tl
3632                 \xglobal \colorlet { nicematrix-last-col } { . }
3633             }
3634         }
3635     }
3636     1
3637     <
3638     {
3639         \@@_math_toggle:
3640         \hbox_set_end:
3641         \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3642         \@@_adjust_size_box:
3643         \@@_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.

```

3644         \dim_gset:Nn \g_@@_width_last_col_dim
3645         { \dim_max:nn { \g_@@_width_last_col_dim } { \box_wd:N \l_@@_cell_box } }
3646         \skip_horizontal:n { -2 \col@sep }

```

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

```

3647 \hbox_overlap_right:n
3648 {
3649   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3650   {
3651     \skip_horizontal:N \l_@@_right_delim_dim
3652     \skip_horizontal:N \l_@@_right_margin_dim
3653     \skip_horizontal:N \l_@@_extra_right_margin_dim
3654     \@@_node_cell:
3655   }
3656 }
3657 \bool_gset_false:N \g_@@_empty_cell_bool
3658 }
3659 }

```

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

```

3660 \NewDocumentEnvironment { NiceArray } { }
3661 {
3662   \bool_gset_false:N \g_@@_delims_bool
3663   \str_if_empty:NT \g_@@_name_env_str
3664   { \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).

```

3665   \NiceArrayWithDelims . .
3666 }
3667 { \endNiceArrayWithDelims }

```

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

```

3668 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3669 {
3670   \NewDocumentEnvironment { #1 NiceArray } { }
3671   {
3672     \bool_gset_true:N \g_@@_delims_bool
3673     \str_if_empty:NT \g_@@_name_env_str
3674     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3675     \@@_test_if_math_mode:
3676     \NiceArrayWithDelims #2 #3
3677   }
3678   { \endNiceArrayWithDelims }
3679 }
3680 \@@_def_env:NNN p ( )
3681 \@@_def_env:NNN b [ ]
3682 \@@_def_env:NNN B \{ \}
3683 \@@_def_env:NNN v \vert \vert
3684 \@@_def_env:NNN V \Vert \Vert

```

13 The environment `{NiceMatrix}` and its variants

```

3685 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3686 {
3687   \bool_set_false:N \l_@@_preamble_bool
3688   \tl_clear:N \l_tmpa_tl
3689   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3690   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3691   \tl_put_right:Nn \l_tmpa_tl

```

```

3692 {
3693   *
3694   {
3695     \int_case:nnF \l_@@_last_col_int
3696     {
3697       { -2 } { \c@MaxMatrixCols }
3698       { -1 } { \int_eval:n { \c@MaxMatrixCols + 1 } } }

```

The value 0 can't occur here since we are in a matrix (which is an environment without preamble).

```

3699   }
3700   { \int_eval:n { \l_@@_last_col_int - 1 } }
3701   }
3702   { #2 }
3703   }
3704   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3705   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3706   }
3707   \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3708   \clist_map_inline:nn { p , b , B , v , V }
3709   {
3710     \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3711     {
3712       \bool_gset_true:N \g_@@_delims_bool
3713       \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3714       \int_if_zero:nT { \l_@@_last_col_int }
3715       {
3716         \bool_set_true:N \l_@@_last_col_without_value_bool
3717         \int_set:Nn \l_@@_last_col_int { -1 }
3718       }
3719       \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3720       \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3721     }
3722     { \use:c { end #1 NiceArray } }
3723   }

```

We define also an environment {NiceMatrix}

```

3724 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3725 {
3726   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3727   \int_if_zero:nT { \l_@@_last_col_int }
3728   {
3729     \bool_set_true:N \l_@@_last_col_without_value_bool
3730     \int_set:Nn \l_@@_last_col_int { -1 }
3731   }
3732   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3733   \bool_lazy_or:nnT
3734   { \clist_if_empty_p:N \l_@@_vlines_clist }
3735   { \l_@@_except_borders_bool }
3736   { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3737   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3738   }
3739   { \endNiceArray }

```

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

```

3740 \cs_new_protected:Npn \@@_NotEmpty:
3741 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3742 \NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
3743 {

```

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`.

```

3744 \dim_compare:nNtT { \l_@@_width_dim } = { \c_zero_dim }
3745 { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3746 \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3747 \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3748 \tl_if_empty:NF \l_@@_short_caption_tl
3749 {
3750   \tl_if_empty:NT \l_@@_caption_tl
3751   {
3752     \@@_error_or_warning:n { short-caption-without~caption }
3753     \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3754   }
3755 }
3756 \tl_if_empty:NF \l_@@_label_tl
3757 {
3758   \tl_if_empty:NT \l_@@_caption_tl
3759   { \@@_error_or_warning:n { label~without~caption } }
3760 }
3761 \NewDocumentEnvironment { TabularNote } { b }
3762 {
3763   \bool_if:NTF \l_@@_in_code_after_bool
3764   { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3765   {
3766     \tl_if_empty:NF \g_@@_tabularnote_tl
3767     { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3768     \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3769   }
3770 }
3771 { }
3772 \@@_settings_for_tabular:
3773 \NiceArray { #2 }
3774 }
3775 { \endNiceArray }
3776 \cs_new_protected:Npn \@@_settings_for_tabular:
3777 {
3778   \bool_set_true:N \l_@@_tabular_bool
3779   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3780   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3781   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3782 }

3783 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3784 {
3785   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3786   \dim_set:Nn \l_@@_width_dim { #1 }
3787   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3788   \@@_settings_for_tabular:
3789   \NiceArray { #3 }
3790 }
3791 {
3792   \endNiceArray
3793   \fp_compare:nNtT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3794   { \@@_error:n { NiceTabularX~without~X } }
3795 }

3796 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3797 {
3798   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3799   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3800   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3801   \@@_settings_for_tabular:

```

```

3802 \NiceArray { #3 }
3803 }
3804 { \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`).

```

3805 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3806 {
3807   \bool_lazy_all:nT
3808   {
3809     { \dim_compare_p:nNn { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim } }
3810     { \l_@@_hvlines_bool }
3811     { ! \g_@@_delims_bool }
3812     { ! \l_@@_except_borders_bool }
3813   }
3814   {
3815     \bool_set_true:N \l_@@_except_borders_bool
3816     \clist_if_empty:NF \l_@@_corners_clist
3817     { \@@_error:n { hvlines,~rounded-corners-and-corners } }
3818     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3819     {
3820       \@@_stroke_block:nnn
3821       {
3822         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3823         draw = \l_@@_rules_color_tl
3824       }
3825       { 1-1 }
3826       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3827     }
3828   }
3829 }

3830 \cs_new_protected:Npn \@@_after_array:
3831 {

```

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`.

```

3832 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3833 \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.

```

3834 \bool_if:NT \g_@@_last_col_found_bool
3835 { \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`.

```

3836 \bool_if:NT \l_@@_last_col_without_value_bool
3837 { \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.

```

3838   \bool_if:NT \l_@@_last_row_without_value_bool
3839   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

3840   \tl_gput_right:Ne \g_@@_aux_tl
3841   {
3842     \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3843     {
3844       \int_use:N \l_@@_first_row_int ,
3845       \int_use:N \c@iRow ,
3846       \int_use:N \g_@@_row_total_int ,
3847       \int_use:N \l_@@_first_col_int ,
3848       \int_use:N \c@jCol ,
3849       \int_use:N \g_@@_col_total_int
3850     }
3851   }

```

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`).

```

3852   \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3853   {
3854     \tl_gput_right:Ne \g_@@_aux_tl
3855     {
3856       \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3857       { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3858     }
3859   }
3860   \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3861   {
3862     \tl_gput_right:Ne \g_@@_aux_tl
3863     {
3864       \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3865       { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3866       \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3867       { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3868     }
3869   }

```

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

```

3870   \@@_create_diag_nodes:

```

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

```

3871   \pgfpicture
3872   \@@_create_aliases_last:
3873   \str_if_empty:NF \l_@@_name_str { \@@_create_alias_nodes: }
3874   \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.

```

3875   \bool_if:NT \l_@@_parallelize_diags_bool
3876   {
3877     \int_gzero:N \g_@@_ddots_int
3878     \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.

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

```

3879     \dim_gzero:N \g_@@_delta_x_one_dim
3880     \dim_gzero:N \g_@@_delta_y_one_dim
3881     \dim_gzero:N \g_@@_delta_x_two_dim
3882     \dim_gzero:N \g_@@_delta_y_two_dim
3883   }
3884   \bool_set_false:N \l_@@_initial_open_bool
3885   \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.

```

3886   \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

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

```

3887   \@@_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.

```

3888   \clist_if_empty:NF \l_@@_corners_clist
3889   {
3890     \bool_if:NTF \l_@@_no_cell_nodes_bool
3891     { \@@_error:n { corners-with-no-cell-nodes } }
3892     { \@@_compute_corners: }
3893   }

```

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

```

3894   \@@_adjust_pos_of_blocks_seq:
3895   \@@_deal_with_rounded_corners:
3896   \clist_if_empty:NF \l_@@_hlines_clist { \@@_draw_hlines: }
3897   \clist_if_empty:NF \l_@@_vlines_clist { \@@_draw_vlines: }

```

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

```

3898   \IfPackageLoadedT { tikz }
3899   {
3900     \tikzset
3901     {
3902       every~picture / .style =
3903       {
3904         overlay ,
3905         remember~picture ,
3906         name~prefix = \@@_env: -
3907       }
3908     }
3909   }
3910   \bool_if:NT \c_@@_recent_array_bool
3911   { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3912   \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3913   \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3914   \cs_set_eq:NN \OverBrace \@@_OverBrace
3915   \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3916   \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3917   \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.

```

3918   \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
3919   \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.

```
3920 \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).

```
3921 \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`).

```
3922 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
3923 { \@@_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:`.

```
3924 \bool_set_true:N \l_@@_in_code_after_bool
3925 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3926 \scan_stop:
3927 \tl_gclear:N \g_nicematrix_code_after_tl
3928 \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.

```
3929 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3930 \tl_if_empty:NF \g_@@_pre_code_before_tl
3931 {
3932   \tl_gput_right:Ne \g_@@_aux_tl
3933   {
3934     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3935     { \exp_not:o \g_@@_pre_code_before_tl }
3936   }
3937   \tl_gclear:N \g_@@_pre_code_before_tl
3938 }
3939 \tl_if_empty:NF \g_nicematrix_code_before_tl
3940 {
3941   \tl_gput_right:Ne \g_@@_aux_tl
3942   {
3943     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3944     { \exp_not:o \g_nicematrix_code_before_tl }
3945   }
3946   \tl_gclear:N \g_nicematrix_code_before_tl
3947 }

3948 \str_gclear:N \g_@@_name_env_str
3949 \@@_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`.

```
3950 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3951 }
```

¹³e.g. `\color[rgb]{0.5,0.5,0}`

```

3952 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
3953 {
3954   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3955   \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }

```

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

```

3956   \dim_set:Nn \l_@@_xdots_shorten_start_dim
3957     { 0.6 \l_@@_xdots_shorten_start_dim }
3958   \dim_set:Nn \l_@@_xdots_shorten_end_dim
3959     { 0.6 \l_@@_xdots_shorten_end_dim }
3960 }

```

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`.

```

3961 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3962 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

```

```

3963 \cs_new_protected:Npn \@@_create_alias_nodes:
3964 {
3965   \int_step_inline:nn { \c@iRow }
3966   {
3967     \pgfnodealias
3968       { \l_@@_name_str - ##1 - last }
3969       { \@@_env: - ##1 - \int_use:N \c@jCol }
3970   }
3971   \int_step_inline:nn { \c@jCol }
3972   {
3973     \pgfnodealias
3974       { \l_@@_name_str - last - ##1 }
3975       { \@@_env: - \int_use:N \c@iRow - ##1 }
3976   }
3977   \pgfnodealias % added 2025-04-05
3978     { \l_@@_name_str - last - last }
3979     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
3980 }

```

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).

```

3981 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3982 {
3983   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3984     { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3985 }

```

The following command must *not* be protected.

```

3986 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3987 {
3988   { #1 }
3989   { #2 }
3990   {
3991     \int_compare:nNnTF { #3 } > { 98 }
3992     { \int_use:N \c@iRow }
3993     { #3 }
3994   }

```

```

3995 {
3996   \int_compare:nNnTF { #4 } > { 98 }
3997     { \int_use:N \c@jCol }
3998     { #4 }
3999 }
4000 { #5 }
4001 }

```

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).

```

4002 \hook_gput_code:nnn { begindocument } { . }
4003 {
4004   \cs_new_protected:Npe \@@_draw_dotted_lines:
4005     {
4006       \c_@@_pgfortikzpicture_tl
4007       \@@_draw_dotted_lines_i:
4008       \c_@@_endpgfortikzpicture_tl
4009     }
4010 }

```

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

```

4011 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4012 {
4013   \pgfrememberpicturepositiononpagetrue
4014   \pgf@relevantforpicturesizefalse
4015   \g_@@_HVdotsfor_lines_tl
4016   \g_@@_Vdots_lines_tl
4017   \g_@@_Ddots_lines_tl
4018   \g_@@_Iddots_lines_tl
4019   \g_@@_Cdots_lines_tl
4020   \g_@@_Ldots_lines_tl
4021 }

4022 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4023 {
4024   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4025   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4026 }

```

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

```

4027 \pgfdeclareshape { @@_diag_node }
4028 {
4029   \savedanchor { \five }
4030   {
4031     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4032     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4033   }
4034   \anchor { 5 } { \five }
4035   \anchor { center } { \pgfpointorigin }
4036   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4037   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4038   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4039   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4040   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4041   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4042   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4043   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4044   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4045   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4046 }

```

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).

```

4047 \cs_new_protected:Npn \@@_create_diag_nodes:
4048 {
4049   \pgfpicture
4050   \pgfrememberpicturepositiononpagetrue
4051   \int_step_inline:nn { \int_max:nn { \c@iRow } { \c@jCol } }
4052   {
4053     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4054     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4055     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4056     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4057     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4058     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4059     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4060     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4061     \pgftransformshift { \pgfpoint \l_tmpa_dim \l_tmpb_dim }

```

Now, `\l_tmpa_dim` and `\l_tmpb_dim` become the width and the height of the node (of shape `@@_diag_node`) that we will construct.

```

4062     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4063     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4064     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4065     \str_if_empty:NF \l_@@_name_str
4066     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4067   }

```

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

```

4068   \int_set:Nn \l_tmpa_int { \int_max:nn { \c@iRow } { \c@jCol } + 1 }
4069   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4070   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4071   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4072   \pgfcoordinate
4073   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4074   \pgfnodealias
4075   { \@@_env: - last }
4076   { \@@_env: - \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
4077   \str_if_empty:NF \l_@@_name_str
4078   {
4079     \pgfnodealias
4080     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4081     { \@@_env: - \int_use:N \l_tmpa_int }
4082     \pgfnodealias
4083     { \l_@@_name_str - last }
4084     { \@@_env: - last }
4085   }
4086   \endpgfpicture
4087 }

```

16 We draw the dotted lines

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

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

The command `\l_@@_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.

```
4088 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4089 {
```

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

```
4090 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4091 \int_set:Nn \l_@@_initial_i_int { #1 }
4092 \int_set:Nn \l_@@_initial_j_int { #2 }
4093 \int_set:Nn \l_@@_final_i_int { #1 }
4094 \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.

```
4095 \bool_set_false:N \l_@@_stop_loop_bool
4096 \bool_do_until:Nn \l_@@_stop_loop_bool
4097 {
4098   \int_add:Nn \l_@@_final_i_int { #3 }
4099   \int_add:Nn \l_@@_final_j_int { #4 }
4100   \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.

```
4101 \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4102   \if_int_compare:w #3 = \c_one_int
4103     \bool_set_true:N \l_@@_final_open_bool
4104   \else:
4105     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4106       \bool_set_true:N \l_@@_final_open_bool
4107     \fi:
4108   \fi:
4109 \else:
4110   \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4111     \if_int_compare:w #4 = -1
4112       \bool_set_true:N \l_@@_final_open_bool
4113     \fi:
4114   \else:
4115     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4116       \if_int_compare:w #4 = \c_one_int
4117         \bool_set_true:N \l_@@_final_open_bool
4118       \fi:
4119     \fi:
4120   \fi:
4121 \fi:
```

```
4122 \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.

```
4123 {
```

We do a step backwards.

```
4124 \int_sub:Nn \l_@@_final_i_int { #3 }
4125 \int_sub:Nn \l_@@_final_j_int { #4 }
4126 \bool_set_true:N \l_@@_stop_loop_bool
4127 }
```

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`.

```
4128 {
4129   \cs_if_exist:cTF
4130   {
4131     @@ _ dotted _
4132     \int_use:N \l_@@_final_i_int -
4133     \int_use:N \l_@@_final_j_int
4134   }
4135   {
4136     \int_sub:Nn \l_@@_final_i_int { #3 }
4137     \int_sub:Nn \l_@@_final_j_int { #4 }
4138     \bool_set_true:N \l_@@_final_open_bool
4139     \bool_set_true:N \l_@@_stop_loop_bool
4140   }
4141   {
4142     \cs_if_exist:cTF
4143     {
4144       pgf @ sh @ ns @ \@@_env:
4145       - \int_use:N \l_@@_final_i_int
4146       - \int_use:N \l_@@_final_j_int
4147     }
4148     { \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.

```
4149 {
4150   \cs_set_nopar:cpn
4151   {
4152     @@ _ dotted _
4153     \int_use:N \l_@@_final_i_int -
4154     \int_use:N \l_@@_final_j_int
4155   }
4156   { }
4157 }
4158 }
4159 }
4160 }
```

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

```
4161 \bool_set_false:N \l_@@_stop_loop_bool
```

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

```
4162 \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4163 \bool_do_until:Nn \l_@@_stop_loop_bool
4164 {
4165   \int_sub:Nn \l_@@_initial_i_int { #3 }
4166   \int_sub:Nn \l_@@_initial_j_int { #4 }
4167   \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.

```

4168     \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4169     \if_int_compare:w #3 = \c_one_int
4170     \bool_set_true:N \l_@@_initial_open_bool
4171     \else:
\l_tmpa_int contains \l_@@_col_min_int - 1 (only for efficiency).
4172     \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4173     \bool_set_true:N \l_@@_initial_open_bool
4174     \fi:
4175     \fi:
4176     \else:
4177     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4178     \if_int_compare:w #4 = \c_one_int
4179     \bool_set_true:N \l_@@_initial_open_bool
4180     \fi:
4181     \else:
4182     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4183     \if_int_compare:w #4 = -1
4184     \bool_set_true:N \l_@@_initial_open_bool
4185     \fi:
4186     \fi:
4187     \fi:
4188     \fi:
4189     \bool_if:NTF \l_@@_initial_open_bool
4190     {
4191     \int_add:Nn \l_@@_initial_i_int { #3 }
4192     \int_add:Nn \l_@@_initial_j_int { #4 }
4193     \bool_set_true:N \l_@@_stop_loop_bool
4194     }
4195     {
4196     \cs_if_exist:cTF
4197     {
4198     @@ _ dotted _
4199     \int_use:N \l_@@_initial_i_int -
4200     \int_use:N \l_@@_initial_j_int
4201     }
4202     {
4203     \int_add:Nn \l_@@_initial_i_int { #3 }
4204     \int_add:Nn \l_@@_initial_j_int { #4 }
4205     \bool_set_true:N \l_@@_initial_open_bool
4206     \bool_set_true:N \l_@@_stop_loop_bool
4207     }
4208     {
4209     \cs_if_exist:cTF
4210     {
4211     pgf @ sh @ ns @ \@@_env:
4212     - \int_use:N \l_@@_initial_i_int
4213     - \int_use:N \l_@@_initial_j_int
4214     }
4215     { \bool_set_true:N \l_@@_stop_loop_bool }
4216     {
4217     \cs_set_nopar:cpn
4218     {
4219     @@ _ dotted _
4220     \int_use:N \l_@@_initial_i_int -
4221     \int_use:N \l_@@_initial_j_int
4222     }
4223     { }
4224     }
4225     }
4226     }

```

```
4227 }
```

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.

```
4228 \seq_gput_right:Np \g_@@_pos_of_xdots_seq
4229 {
4230   { \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`.

```
4231   { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4232   { \int_use:N \l_@@_final_i_int }
4233   { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4234   { } % for the name of the block
4235 }
4236 }
```

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

```
4237 \cs_new_protected:Npn \@@_open_shorten:
4238 {
4239   \bool_if:NT \l_@@_initial_open_bool
4240     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4241   \bool_if:NT \l_@@_final_open_bool
4242     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4243 }
```

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).

```
4244 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4245 {
4246   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4247   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4248   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4249   \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`.

```
4250 \seq_if_empty:NF \g_@@_submatrix_seq
4251 {
4252   \seq_map_inline:Nn \g_@@_submatrix_seq
4253     { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
4254 }
4255 }
```

#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.

```

4256 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4257 {
4258   \if_int_compare:w #3 > #1
4259   \else:
4260     \if_int_compare:w #1 > #5
4261     \else:
4262       \if_int_compare:w #4 > #2
4263       \else:
4264         \if_int_compare:w #2 > #6
4265         \else:
4266           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4267           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4268           \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4269           \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4270         \fi:
4271       \fi:
4272     \fi:
4273   \fi:
4274 }

4275 \cs_new_protected:Npn \@@_set_initial_coords:
4276 {
4277   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4278   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4279 }
4280 \cs_new_protected:Npn \@@_set_final_coords:
4281 {
4282   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4283   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4284 }
4285 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4286 {
4287   \pgfpointanchor
4288   {
4289     \@@_env:
4290     - \int_use:N \l_@@_initial_i_int
4291     - \int_use:N \l_@@_initial_j_int
4292   }
4293   { #1 }
4294   \@@_set_initial_coords:
4295 }
4296 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4297 {
4298   \pgfpointanchor
4299   {
4300     \@@_env:
4301     - \int_use:N \l_@@_final_i_int
4302     - \int_use:N \l_@@_final_j_int
4303   }
4304   { #1 }
4305   \@@_set_final_coords:
4306 }

```

```

4307 \cs_new_protected:Npn \@@_open_x_initial_dim:
4308 {
4309   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4310   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4311   {
4312     \cs_if_exist:cT
4313     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4314     {
4315       \pgfpointanchor
4316       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4317       { west }
4318       \dim_set:Nn \l_@@_x_initial_dim
4319       { \dim_min:nn { \l_@@_x_initial_dim } { \pgf@x } }
4320     }
4321   }

```

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

```

4322   \dim_compare:nNnT { \l_@@_x_initial_dim } = { \c_max_dim }
4323   {
4324     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4325     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4326     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4327   }
4328 }

4329 \cs_new_protected:Npn \@@_open_x_final_dim:
4330 {
4331   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4332   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4333   {
4334     \cs_if_exist:cT
4335     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4336     {
4337       \pgfpointanchor
4338       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4339       { east }
4340       \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
4341       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4342     }
4343   }

```

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

```

4344   \dim_compare:nNnT { \l_@@_x_final_dim } = { - \c_max_dim }
4345   {
4346     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4347     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4348     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4349   }
4350 }

```

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.

```

4351 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4352 {
4353   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4354   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4355   {
4356     \@@_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.

```

4357   \group_begin:
4358   \@@_open_shorten:

```

```

4359     \int_if_zero:nTF { #1 }
4360     { \color { nicematrix-first-row } }
4361     {

```

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.

```

4362         \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4363         { \color { nicematrix-last-row } }
4364     }
4365     \keys_set:nn { nicematrix / xdots } { #3 }
4366     \@@_color:o \l_@@_xdots_color_tl
4367     \@@_actually_draw_Ldots:
4368     \group_end:
4369 }
4370 }

```

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`.

```

4371 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4372 {
4373     \bool_if:NTF \l_@@_initial_open_bool
4374     {
4375         \@@_open_x_initial_dim:
4376         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4377         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4378     }
4379     { \@@_set_initial_coords_from_anchor:n { base-east } }
4380     \bool_if:NTF \l_@@_final_open_bool
4381     {
4382         \@@_open_x_final_dim:
4383         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4384         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4385     }
4386     { \@@_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.

```

4387     \bool_lazy_all:nTF
4388     {
4389         \l_@@_initial_open_bool
4390         \l_@@_final_open_bool
4391         { \int_compare_p:nNn { \l_@@_initial_i_int } = { \l_@@_last_row_int } }
4392     }
4393     {
4394         \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4395         \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4396     }

```

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 (?).

```

4397     {
4398         \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim

```

```

4399     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4400   }
4401   \@@_draw_line:
4402 }

```

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.

```

4403 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4404 {
4405   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4406   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4407   {
4408     \@@_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.

```

4409     \group_begin:
4410     \@@_open_shorten:
4411     \int_if_zero:nTF { #1 }
4412     { \color { nicematrix-first-row } }
4413     {

```

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.

```

4414         \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4415         { \color { nicematrix-last-row } }
4416     }
4417     \keys_set:nn { nicematrix / xdots } { #3 }
4418     \@@_color:o \l_@@_xdots_color_tl
4419     \@@_actually_draw_Cdots:
4420   \group_end:
4421 }
4422 }

```

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`.

```

4423 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4424 {
4425   \bool_if:NTF \l_@@_initial_open_bool
4426   { \@@_open_x_initial_dim: }
4427   { \@@_set_initial_coords_from_anchor:n { mid-east } }
4428   \bool_if:NTF \l_@@_final_open_bool
4429   { \@@_open_x_final_dim: }
4430   { \@@_set_final_coords_from_anchor:n { mid-west } }
4431   \bool_lazy_and:nnTF
4432   { \l_@@_initial_open_bool }
4433   { \l_@@_final_open_bool }
4434   {
4435     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4436     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4437     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4438     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }

```

```

4439     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4440   }
4441   {
4442     \bool_if:NT \l_@@_initial_open_bool
4443     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4444     \bool_if:NT \l_@@_final_open_bool
4445     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4446   }
4447   \@@_draw_line:
4448 }
4449 \cs_new_protected:Npn \@@_open_y_initial_dim:
4450 {
4451   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4452   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4453   {
4454     \cs_if_exist:cT
4455     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4456     {
4457       \pgfpointanchor
4458       { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4459       { north }
4460       \dim_compare:nNnT { \pgf@y } > { \l_@@_y_initial_dim }
4461       { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4462     }
4463   }
4464   \dim_compare:nNnT { \l_@@_y_initial_dim } = { - \c_max_dim }
4465   {
4466     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4467     \dim_set:Nn \l_@@_y_initial_dim
4468     {
4469       \fp_to_dim:n
4470       {
4471         \pgf@y
4472         + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4473       }
4474     }
4475   }
4476 }
4477 \cs_new_protected:Npn \@@_open_y_final_dim:
4478 {
4479   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4480   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4481   {
4482     \cs_if_exist:cT
4483     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4484     {
4485       \pgfpointanchor
4486       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4487       { south }
4488       \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
4489       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4490     }
4491   }
4492   \dim_compare:nNnT { \l_@@_y_final_dim } = { \c_max_dim }
4493   {
4494     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4495     \dim_set:Nn \l_@@_y_final_dim
4496     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4497   }
4498 }

```

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.

```

4499 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4500 {
4501   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4502   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4503   {
4504     \@@_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.

```

4505     \group_begin:
4506     \@@_open_shorten:
4507     \int_if_zero:nTF { #2 }
4508     { \color { nicematrix-first-col } }
4509     {
4510       \int_compare:nNtT { #2 } = { \l_@@_last_col_int }
4511       { \color { nicematrix-last-col } }
4512     }
4513     \keys_set:nn { nicematrix / xdots } { #3 }
4514     \@@_color:o \l_@@_xdots_color_tl
4515     \bool_if:NTF \l_@@_Vbrace_bool
4516     { \@@_actually_draw_Vbrace: }
4517     { \@@_actually_draw_Vdots: }
4518   \group_end:
4519 }
4520 }

```

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`.

```

4521 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4522 {
4523   \bool_lazy_and:nnTF { \l_@@_initial_open_bool } { \l_@@_final_open_bool }
4524   { \@@_actually_draw_Vdots_i: }
4525   { \@@_actually_draw_Vdots_ii: }
4526   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4527   \@@_draw_line:
4528 }

```

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

```

4529 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4530 {
4531   \@@_open_y_initial_dim:
4532   \@@_open_y_final_dim:
4533   \int_if_zero:nTF { \l_@@_initial_j_int }

```

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

```

4534   {
4535     \@@_qpoint:n { col - 1 }
4536     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4537     \dim_sub:Nn \l_@@_x_initial_dim
4538     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4539   }
4540   {

```



```

4541 \bool_lazy_and:nnTF
4542 { \int_compare_p:nNn { \l_@@_last_col_int } > { -2 } }
4543 {
4544   \int_compare_p:nNn
4545   { \l_@@_initial_j_int } = { \g_@@_col_total_int }
4546 }

```

We have a dotted line open on both sides and which is in the “last column”.

```

4547 {
4548   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4549   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4550   \dim_add:Nn \l_@@_x_initial_dim
4551   { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4552 }

```

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

```

4553 {
4554   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4555   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4556   \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4557   \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4558 }
4559 }
4560 }

```

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.

```

4561 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4562 {
4563   \bool_set_false:N \l_tmpa_bool
4564   \bool_if:NF \l_@@_initial_open_bool
4565   {
4566     \bool_if:NF \l_@@_final_open_bool
4567     {
4568       \@@_set_initial_coords_from_anchor:n { south~west }
4569       \@@_set_final_coords_from_anchor:n { north~west }
4570       \bool_set:Nn \l_tmpa_bool
4571       {
4572         \dim_compare_p:nNn
4573         { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4574       }
4575     }
4576   }

```

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

```

4577 \bool_if:NTF \l_@@_initial_open_bool
4578 {
4579   \@@_open_y_initial_dim:
4580   \@@_set_final_coords_from_anchor:n { north }
4581   \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4582 }
4583 {
4584   \@@_set_initial_coords_from_anchor:n { south }
4585   \bool_if:NTF \l_@@_final_open_bool
4586   { \@@_open_y_final_dim: }

```

Now the case where both extremities are closed. The first conditional tests whether the column is of type `c` or may be considered as if.

```

4587 {
4588   \@@_set_final_coords_from_anchor:n { north }
4589   \dim_compare:nNnF { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4590   {

```

```

4591         \dim_set:Nn \l_@@_x_initial_dim
4592         {
4593             \bool_if:NTF \l_tmpa_bool { \dim_min:nn } { \dim_max:nn }
4594             \l_@@_x_initial_dim \l_@@_x_final_dim
4595         }
4596     }
4597 }
4598 }
4599 }

```

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`.

```

4600 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4601 {
4602     \bool_if:NTF \l_@@_initial_open_bool
4603     { \@@_open_y_initial_dim: }
4604     { \@@_set_initial_coords_from_anchor:n { south } }
4605     \bool_if:NTF \l_@@_final_open_bool
4606     { \@@_open_y_final_dim: }
4607     { \@@_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.

```

4608     \int_if_zero:nTF { \l_@@_initial_j_int }
4609     {
4610         \@@_qpoint:n { col - 1 }
4611         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4612         \dim_sub:Nn \l_@@_x_initial_dim
4613         { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4614     }

```

Elsewhere, the brace must be drawn left flush.

```

4615     {
4616         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4617         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4618         \dim_add:Nn \l_@@_x_initial_dim
4619         { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4620     }

```

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

```

4621     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4622     \@@_draw_line:
4623 }

```

```

4624 \cs_new:Npn \@@_colsep:
4625 { \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.

```

4626 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4627 {
4628   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4629   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4630   {
4631     \@@_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.

```

4632   \group_begin:
4633   \@@_open_shorten:
4634   \keys_set:nn { nicematrix / xdots } { #3 }
4635   \@@_color:o \l_@@_xdots_color_tl
4636   \@@_actually_draw_Ddots:
4637   \group_end:
4638 }
4639 }

```

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`.

```

4640 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4641 {
4642   \bool_if:NTF \l_@@_initial_open_bool
4643   {
4644     \@@_open_y_initial_dim:
4645     \@@_open_x_initial_dim:
4646   }
4647   { \@@_set_initial_coords_from_anchor:n { south-east } }
4648   \bool_if:NTF \l_@@_final_open_bool
4649   {
4650     \@@_open_x_final_dim:
4651     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4652   }
4653   { \@@_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.

```

4654   \bool_if:NT \l_@@_parallelize_diags_bool
4655   {
4656     \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).

```

4657     \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.

```

4658     {

```

```

4659         \dim_gset:Nn \g_@@_delta_x_one_dim
4660         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4661         \dim_gset:Nn \g_@@_delta_y_one_dim
4662         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4663     }

```

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`.

```

4664     {
4665         \dim_compare:nNnF { \g_@@_delta_x_one_dim } = { \c_zero_dim }
4666         {
4667             \dim_set:Nn \l_@@_y_final_dim
4668             {
4669                 \l_@@_y_initial_dim +
4670                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4671                 \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4672             }
4673         }
4674     }
4675 }
4676 \@@_draw_line:
4677 }

```

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.

```

4678 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4679 {
4680     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4681     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4682     {
4683         \@@_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.

```

4684     \group_begin:
4685     \@@_open_shorten:
4686     \keys_set:nn { nicematrix / xdots } { #3 }
4687     \@@_color:o \l_@@_xdots_color_tl
4688     \@@_actually_draw_Iddots:
4689     \group_end:
4690 }
4691 }

```

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`.

```

4692 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4693 {
4694     \bool_if:NTF \l_@@_initial_open_bool
4695     {
4696         \@@_open_y_initial_dim:
4697         \@@_open_x_initial_dim:
4698     }

```

```

4699     { \@@_set_initial_coords_from_anchor:n { south-west } }
4700 \bool_if:NTF \l_@@_final_open_bool
4701 {
4702     \@@_open_y_final_dim:
4703     \@@_open_x_final_dim:
4704 }
4705 { \@@_set_final_coords_from_anchor:n { north-east } }
4706 \bool_if:NT \l_@@_parallelize_diags_bool
4707 {
4708     \int_gincr:N \g_@@_iddots_int
4709     \int_compare:nNnTF { \g_@@_iddots_int } = { \c_one_int }
4710     {
4711         \dim_gset:Nn \g_@@_delta_x_two_dim
4712         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4713         \dim_gset:Nn \g_@@_delta_y_two_dim
4714         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4715     }
4716     {
4717         \dim_compare:nNnF { \g_@@_delta_x_two_dim } = { \c_zero_dim }
4718         {
4719             \dim_set:Nn \l_@@_y_final_dim
4720             {
4721                 \l_@@_y_initial_dim +
4722                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4723                 \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4724             }
4725         }
4726     }
4727 }
4728 \@@_draw_line:
4729 }

```

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`

```

4730 \cs_new_protected:Npn \@@_draw_line:
4731 {
4732     \pgfrememberpicturepositiononpagetrue
4733     \pgf@relevantforpicturesizefalse
4734     \bool_lazy_or:nnTF
4735     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4736     { \l_@@_dotted_bool }
4737     { \@@_draw_standard_dotted_line: }
4738     { \@@_draw_unstandard_dotted_line: }
4739 }

```

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.

```

4740 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4741 {
4742   \begin { scope }
4743   \@@_draw_unstandard_dotted_line:o
4744   { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4745 }

```

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.

```

4746 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4747 {
4748   \@@_draw_unstandard_dotted_line:nooo
4749   { #1 }
4750   \l_@@_xdots_up_tl
4751   \l_@@_xdots_down_tl
4752   \l_@@_xdots_middle_tl
4753 }
4754 \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.

```

4755 \hook_gput_code:nnn { begindocument } { . }
4756 {
4757   \IfPackageLoadedT { tikz }
4758   {
4759     \tikzset
4760     {
4761       @@_node_above / .style = { sloped , above } ,
4762       @@_node_below / .style = { sloped , below } ,
4763       @@_node_middle / .style =
4764       {
4765         sloped ,
4766         inner~sep = \c_@@_innersep_middle_dim
4767       }
4768     }
4769   }
4770 }

4771 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4772 {

```

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.

```

4773 \dim_zero_new:N \l_@@_l_dim
4774 \dim_set:Nn \l_@@_l_dim
4775 {
4776   \fp_to_dim:n
4777   {
4778     sqrt
4779     (
4780       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4781       +
4782       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4783     )
4784   }
4785 }

```

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.

```

4786   \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4787   {
4788     \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4789     \@@_draw_unstandard_dotted_line_i:
4790   }

```

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

```

4791   \bool_if:NT \l_@@_xdots_h_labels_bool
4792   {
4793     \tikzset
4794     {
4795       @@_node_above / .style = { auto = left } ,
4796       @@_node_below / .style = { auto = right } ,
4797       @@_node_middle / .style = { innersep = \c_@@_innersep_middle_dim }
4798     }
4799   }
4800   \tl_if_empty:nF { #4 }
4801   { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4802   \draw
4803   [ #1 ]
4804   ( \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).

```

4805     -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4806     node [ @@_node_below ] { $ \scriptstyle #3 $ }
4807     node [ @@_node_above ] { $ \scriptstyle #2 $ }
4808     ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4809   \end { scope }
4810 }
4811 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4812 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4813 {
4814   \dim_set:Nn \l_tmpa_dim
4815   {
4816     \l_@@_x_initial_dim
4817     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4818     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4819   }
4820   \dim_set:Nn \l_tmpb_dim
4821   {
4822     \l_@@_y_initial_dim
4823     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4824     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4825   }
4826   \dim_set:Nn \l_@@_tmpc_dim
4827   {
4828     \l_@@_x_final_dim
4829     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4830     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4831   }
4832   \dim_set:Nn \l_@@_tmpd_dim
4833   {
4834     \l_@@_y_final_dim
4835     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4836     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4837   }
4838   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4839   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim

```

```

4840 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4841 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4842 }

```

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

```

4843 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4844 {
4845   \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.

```

4846   \dim_zero_new:N \l_@@_l_dim
4847   \dim_set:Nn \l_@@_l_dim
4848   {
4849     \fp_to_dim:n
4850     {
4851       sqrt
4852       (
4853         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4854         +
4855         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4856       )
4857     }
4858   }

```

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.

```

4859   \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4860   {
4861     \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4862     { \@@_draw_standard_dotted_line_i: }
4863   }
4864   \group_end:
4865   \bool_lazy_all:nF
4866   {
4867     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4868     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4869     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4870   }
4871   { \@@_labels_standard_dotted_line: }
4872 }
4873 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4874 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4875 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

4876   \int_set:Nn \l_tmpa_int
4877   {
4878     \dim_ratio:nn
4879     {
4880       \l_@@_l_dim
4881       - \l_@@_xdots_shorten_start_dim
4882       - \l_@@_xdots_shorten_end_dim
4883     }
4884     { \l_@@_xdots_inter_dim }
4885   }

```


The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

4886   \dim_set:Nn \l_tmpa_dim
4887   {
4888     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4889     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4890   }
4891   \dim_set:Nn \l_tmpb_dim
4892   {
4893     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4894     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4895   }

```

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.

```

4896   \dim_gadd:Nn \l_@@_x_initial_dim
4897   {
4898     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4899     \dim_ratio:nn
4900     {
4901       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4902       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4903     }
4904     { 2 \l_@@_l_dim }
4905   }
4906   \dim_gadd:Nn \l_@@_y_initial_dim
4907   {
4908     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4909     \dim_ratio:nn
4910     {
4911       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4912       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4913     }
4914     { 2 \l_@@_l_dim }
4915   }
4916   \pgf@relevantforpicturesizefalse
4917   \int_step_inline:nnn { \c_zero_int } { \l_tmpa_int }
4918   {
4919     \pgfpathcircle
4920     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4921     { \l_@@_xdots_radius_dim }
4922     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4923     \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4924   }
4925   \pgfusepathqfill
4926 }

```

```

4927 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
4928 {
4929   \pgfscope
4930   \pgftransformshift
4931   {
4932     \pgfpointlineattime { 0.5 }
4933     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4934     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4935   }
4936   \fp_set:Nn \l_tmpa_fp
4937   {
4938     atand
4939     (
4940       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4941       \l_@@_x_final_dim - \l_@@_x_initial_dim
4942     )

```

```

4943     }
4944     \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4945     \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4946     \tl_if_empty:NF \l_@@_xdots_middle_tl
4947     {
4948         \begin { pgfscope }
4949         \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4950         \pgfnode
4951         { rectangle }
4952         { center }
4953         {
4954             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4955             {
4956                 \c_math_toggle_token
4957                 \scriptstyle \l_@@_xdots_middle_tl
4958                 \c_math_toggle_token
4959             }
4960         }
4961         { }
4962         {
4963             \pgfsetfillcolor { white }
4964             \pgfusepath { fill }
4965         }
4966         \end { pgfscope }
4967     }
4968     \tl_if_empty:NF \l_@@_xdots_up_tl
4969     {
4970         \pgfnode
4971         { rectangle }
4972         { south }
4973         {
4974             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4975             {
4976                 \c_math_toggle_token
4977                 \scriptstyle \l_@@_xdots_up_tl
4978                 \c_math_toggle_token
4979             }
4980         }
4981         { }
4982         { \pgfusepath { } }
4983     }
4984     \tl_if_empty:NF \l_@@_xdots_down_tl
4985     {
4986         \pgfnode
4987         { rectangle }
4988         { north }
4989         {
4990             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4991             {
4992                 \c_math_toggle_token
4993                 \scriptstyle \l_@@_xdots_down_tl
4994                 \c_math_toggle_token
4995             }
4996         }
4997         { }
4998         { \pgfusepath { } }
4999     }
5000     \endpgfscope
5001 }

```

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.

```
5002 \hook_gput_code:nnn { begindocument } { . }
5003 {
```

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

```
5004 \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
5005 \cs_new_protected:Npn \@@_Ldots:
5006 { \@@_collect_options:n { \@@_Ldots_i } }
5007 \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5008 {
5009   \int_if_zero:nTF { \c@jCol }
5010   { \@@_error:nn { in~first~col } { \Ldots } }
5011   {
5012     \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5013     { \@@_error:nn { in~last~col } { \Ldots } }
5014     {
5015       \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5016       { #1 , down = #2 , up = #3 , middle = #4 }
5017     }
5018   }
5019   \bool_if:NF \l_@@_nullify_dots_bool
5020   { \phantom { \ensuremath { \@@_old_ldots: } } }
5021   \bool_gset_true:N \g_@@_empty_cell_bool
5022 }
```

```
5023 \cs_new_protected:Npn \@@_Cdots:
5024 { \@@_collect_options:n { \@@_Cdots_i } }
5025 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5026 {
5027   \int_if_zero:nTF { \c@jCol }
5028   { \@@_error:nn { in~first~col } { \Cdots } }
5029   {
5030     \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5031     { \@@_error:nn { in~last~col } { \Cdots } }
5032     {
5033       \@@_instruction_of_type:nnn { \c_false_bool } { \Cdots }
5034       { #1 , down = #2 , up = #3 , middle = #4 }
5035     }
5036   }
5037   \bool_if:NF \l_@@_nullify_dots_bool
5038   { \phantom { \ensuremath { \@@_old_cdots: } } }
5039   \bool_gset_true:N \g_@@_empty_cell_bool
5040 }
```

```
5041 \cs_new_protected:Npn \@@_Vdots:
5042 { \@@_collect_options:n { \@@_Vdots_i } }
5043 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5044 {
5045   \int_if_zero:nTF { \c@iRow }
```

```

5046 { \@@_error:nn { in~first~row } { \Vdots } }
5047 {
5048   \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
5049   { \@@_error:nn { in~last~row } { \Vdots } }
5050   {
5051     \@@_instruction_of_type:nnn { \c_false_bool } { \Vdots }
5052     { #1 , down = #2 , up = #3 , middle = #4 }
5053   }
5054 }
5055 \bool_if:NF \l_@@_nullify_dots_bool
5056 { \phantom { \ensuremath { \@@_old_vdots: } } } }
5057 \bool_gset_true:N \g_@@_empty_cell_bool
5058 }

5059 \cs_new_protected:Npn \@@_Ddots:
5060 { \@@_collect_options:n { \@@_Ddots_i } }
5061 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5062 {
5063   \int_case:nnF \c@iRow
5064   {
5065     0 { \@@_error:nn { in~first~row } { \Ddots } }
5066     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5067   }
5068   {
5069     \int_case:nnF \c@jCol
5070     {
5071       0 { \@@_error:nn { in~first~col } { \Ddots } }
5072       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5073     }
5074     {
5075       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5076       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
5077       { #1 , down = #2 , up = #3 , middle = #4 }
5078     }
5079   }
5080 }
5081 \bool_if:NF \l_@@_nullify_dots_bool
5082 { \phantom { \ensuremath { \@@_old_ddots: } } } }
5083 \bool_gset_true:N \g_@@_empty_cell_bool
5084 }

5085 \cs_new_protected:Npn \@@_Iddots:
5086 { \@@_collect_options:n { \@@_Iddots_i } }
5087 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5088 {
5089   \int_case:nnF \c@iRow
5090   {
5091     0 { \@@_error:nn { in~first~row } { \Iddots } }
5092     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5093   }
5094   {
5095     \int_case:nnF \c@jCol
5096     {
5097       0 { \@@_error:nn { in~first~col } { \Iddots } }
5098       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5099     }
5100     {
5101       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5102       \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { \Iddots }
5103       { #1 , down = #2 , up = #3 , middle = #4 }
5104     }
5105   }

```

```

5106      \bool_if:NF \l_@@_nullify_dots_bool
5107      { \phantom { \ensuremath { \@@_old_iddots: } } }
5108      \bool_gset_true:N \g_@@_empty_cell_bool
5109    }
5110  }

```

End of the `\AddToHook`.

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

```

5111 \keys_define:nn { nicematrix / Ddots }
5112 {
5113   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5114   draw-first .default:n = true ,
5115   draw-first .value_forbidden:n = true
5116 }

```

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

```

5117 \cs_new_protected:Npn \@@_Hspace:
5118 {
5119   \bool_gset_true:N \g_@@_empty_cell_bool
5120   \hspace
5121 }

```

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`.

```

5122 \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`.

```

5123 \cs_new:Npn \@@_Hdotsfor:
5124 {
5125   \bool_lazy_and:nnTF
5126     { \int_if_zero_p:n { \c@jCol } }
5127     { \int_if_zero_p:n { \l_@@_first_col_int } }
5128     {
5129       \bool_if:NTF \g_@@_after_col_zero_bool
5130       {
5131         \multicolumn { 1 } { c } { }
5132         \@@_Hdotsfor_i:
5133       }
5134       { \@@_fatal:n { Hdotsfor~in~col~0 } }
5135     }
5136   {
5137     \multicolumn { 1 } { c } { }
5138     \@@_Hdotsfor_i:
5139   }
5140 }

```

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:`).

```

5141 \hook_gput_code:nnn { begindocument } { . }
5142 {

```

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

```

5143   \cs_new_protected:Npn \@@_Hdotsfor_i:
5144   { \@@_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`).

```

5145 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5146 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5147 {
5148   \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5149   {
5150     \@@_Hdotsfor:nnnn
5151     { \int_use:N \c@iRow }
5152     { \int_use:N \c@jCol }
5153     { #2 }
5154     {
5155       #1 , #3 ,
5156       down = \exp_not:n { #4 } ,
5157       up = \exp_not:n { #5 } ,
5158       middle = \exp_not:n { #6 }
5159     }
5160   }
5161   \prg_replicate:nn { #2 - 1 }
5162   {
5163     &
5164     \multicolumn { 1 } { c } { }
5165     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5166   }
5167 }
5168 }

```

```

5169 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5170 {
5171   \bool_set_false:N \l_@@_initial_open_bool
5172   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5173 \int_set:Nn \l_@@_initial_i_int { #1 }
5174 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5175 \int_compare:nNnTF { #2 } = { \c_one_int }
5176 {
5177   \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5178   \bool_set_true:N \l_@@_initial_open_bool
5179 }
5180 {
5181   \cs_if_exist:cTF
5182   {
5183     pgf @ sh @ ns @ \@@_env:
5184     - \int_use:N \l_@@_initial_i_int
5185     - \int_eval:n { #2 - 1 }
5186   }
5187   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5188   {
5189     \int_set:Nn \l_@@_initial_j_int { #2 }
5190     \bool_set_true:N \l_@@_initial_open_bool
5191   }
5192 }
5193 \int_compare:nNnTF { #2 + #3 - 1 } = { \c@jCol }
5194 {
5195   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5196   \bool_set_true:N \l_@@_final_open_bool
5197 }
5198 {
5199   \cs_if_exist:cTF
5200   {
5201     pgf @ sh @ ns @ \@@_env:

```

```

5202         - \int_use:N \l_@@_final_i_int
5203         - \int_eval:n { #2 + #3 }
5204     }
5205     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5206     {
5207         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5208         \bool_set_true:N \l_@@_final_open_bool
5209     }
5210 }
5211 \group_begin:
5212 \@@_open_shorten:
5213 \int_if_zero:nTF { #1 }
5214 { \color { nicematrix-first-row } }
5215 {
5216     \int_compare:nNnT { #1 } = { \g_@@_row_total_int }
5217     { \color { nicematrix-last-row } }
5218 }
5219 \keys_set:nn { nicematrix / xdots } { #4 }
5220 \@@_color:o \l_@@_xdots_color_tl
5221 \@@_actually_draw_Ldots:
5222 \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).

```

5223 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5224 { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5225 }

```

```

5226 \hook_gput_code:nnn { begindocument } { . }
5227 {
5228     \cs_new_protected:Npn \@@_Vdotsfor:
5229     { \@@_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`).

```

5230 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5231 \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5232 {
5233     \bool_gset_true:N \g_@@_empty_cell_bool
5234     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5235     {
5236         \@@_Vdotsfor:nnnn
5237         { \int_use:N \c@iRow }
5238         { \int_use:N \c@jCol }
5239         { #2 }
5240         {
5241             #1 , #3 ,
5242             down = \exp_not:n { #4 } ,
5243             up = \exp_not:n { #5 } ,
5244             middle = \exp_not:n { #6 }
5245         }
5246     }
5247 }
5248 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5249 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5250 {
5251     \bool_set_false:N \l_@@_initial_open_bool
5252     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```
5253 \int_set:Nn \l_@@_initial_j_int { #2 }
5254 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int
```

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

```
5255 \int_compare:nNnTF { #1 } = { \c_one_int }
5256 {
5257   \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5258   \bool_set_true:N \l_@@_initial_open_bool
5259 }
5260 {
5261   \cs_if_exist:cTF
5262   {
5263     pgf @ sh @ ns @ \@@_env:
5264     - \int_eval:n { #1 - 1 }
5265     - \int_use:N \l_@@_initial_j_int
5266   }
5267   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5268   {
5269     \int_set:Nn \l_@@_initial_i_int { #1 }
5270     \bool_set_true:N \l_@@_initial_open_bool
5271   }
5272 }
5273 \int_compare:nNnTF { #1 + #3 - 1 } = { \c@iRow }
5274 {
5275   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5276   \bool_set_true:N \l_@@_final_open_bool
5277 }
5278 {
5279   \cs_if_exist:cTF
5280   {
5281     pgf @ sh @ ns @ \@@_env:
5282     - \int_eval:n { #1 + #3 }
5283     - \int_use:N \l_@@_final_j_int
5284   }
5285   { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5286   {
5287     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5288     \bool_set_true:N \l_@@_final_open_bool
5289   }
5290 }
5291 \group_begin:
5292 \@@_open_shorten:
5293 \int_if_zero:nTF { #2 }
5294 { \color { nicematrix-first-col } }
5295 {
5296   \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
5297   { \color { nicematrix-last-col } }
5298 }
5299 \keys_set:nn { nicematrix / xdots } { #4 }
5300 \@@_color:o \l_@@_xdots_color_tl
5301 \bool_if:NTF \l_@@_Vbrace_bool
5302 { \@@_actually_draw_Vbrace: }
5303 { \@@_actually_draw_Vdots: }
5304 \group_end:

5305 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5306 { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5307 }
```

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}`.

```

5308 \NewDocumentCommand \@@_rotate: { 0 { } }
5309 {
5310   \bool_gset_true:N \g_@@_rotate_bool
5311   \keys_set:nn { nicematrix / rotate } { #1 }
5312   \ignorespaces
5313 }

5314 \keys_define:nn { nicematrix / rotate }
5315 {
5316   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5317   c .value_forbidden:n = true ,
5318   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5319 }

```

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

```

5320 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5321 {
5322   \tl_if_empty:nTF { #2 }
5323     { #1 }
5324     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5325 }
5326 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5327 { \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).

```

5328 \hook_gput_code:nnn { begindocument } { . }
5329 {

```

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

```

5330   \tl_set_rescan:Nnn \l_tmpa_tl { }
5331   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5332   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5333   {
5334     \group_begin:
5335     \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5336     \@@_color:o \l_@@_xdots_color_tl
5337     \use:e
5338     {

```

¹⁴Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

5339         \@@_line_i:nn
5340         { \@@_double_int_eval:n #2 - \q_stop }
5341         { \@@_double_int_eval:n #3 - \q_stop }
5342     }
5343     \group_end:
5344 }
5345 }

5346 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5347 {
5348     \bool_set_false:N \l_@@_initial_open_bool
5349     \bool_set_false:N \l_@@_final_open_bool
5350     \bool_lazy_or:nnTF
5351     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5352     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5353     { \@@_error:nnn { unknown-cell-for-line-in~CodeAfter } { #1 } { #2 } }

```

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

```

5354     { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5355 }

5356 \hook_gput_code:nnn { begindocument } { . }
5357 {
5358     \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5359     {

```

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:`.

```

5360     \c_@@_pgfortikzpicture_tl
5361     \@@_draw_line_iii:nn { #1 } { #2 }
5362     \c_@@_endpgfortikzpicture_tl
5363 }
5364 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5365 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5366 {
5367     \pgfrememberpicturepositiononpagetrue
5368     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5369     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5370     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5371     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5372     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5373     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5374     \@@_draw_line:
5375 }

```

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.

```
5376 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNnT { \c@iRow } < }
5377 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNnF { \c@jCol } < }
```

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

```
5378 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5379 {
5380   \tl_gput_right:Ne \g_@@_row_style_tl
5381   {
```

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

```
5382     \exp_not:N
5383     \@@_if_row_less_than:nn
5384     { \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`).

```
5385     {
5386       \exp_not:N
5387       \@@_if_col_greater_than:nn
5388       { \int_eval:n { \c@jCol } }
5389       { \exp_not:n { #1 } \scan_stop: }
5390     }
5391   }
5392 }
5393 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
```

```
5394 \keys_define:nn { nicematrix / RowStyle }
5395 {
5396   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5397   cell-space-top-limit .value_required:n = true ,
5398   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5399   cell-space-bottom-limit .value_required:n = true ,
5400   cell-space-limits .meta:n =
5401   {
5402     cell-space-top-limit = #1 ,
5403     cell-space-bottom-limit = #1 ,
5404   } ,
5405   color .tl_set:N = \l_@@_color_tl ,
5406   color .value_required:n = true ,
5407   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5408   bold .default:n = true ,
5409   nb-rows .code:n =
5410   \str_if_eq:eeTF { #1 } { * }
5411   { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5412   { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5413   nb-rows .value_required:n = true ,
5414   fill .tl_set:N = \l_@@_fill_tl ,
5415   fill .value_required:n = true ,
5416   opacity .tl_set:N = \l_@@_opacity_tl ,
5417   opacity .value_required:n = true ,
5418   rowcolor .tl_set:N = \l_@@_fill_tl ,
5419   rowcolor .value_required:n = true ,
5420   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5421   rounded-corners .default:n = 4 pt ,
5422   unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5423 }
```

```

5424 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5425 {
5426   \group_begin:
5427   \tl_clear:N \l_@@_fill_tl
5428   \tl_clear:N \l_@@_opacity_tl
5429   \tl_clear:N \l_@@_color_tl
5430   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5431   \dim_zero:N \l_@@_rounded_corners_dim
5432   \dim_zero:N \l_tmpa_dim
5433   \dim_zero:N \l_tmpb_dim
5434   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5435   \tl_if_empty:NF \l_@@_fill_tl
5436   {
5437     \@@_add_opacity_to_fill:
5438     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5439     {

```

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

```

5440       \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5441       { \int_use:N \c@iRow - \int_use:N \c@jCol }
5442       {
5443         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5444         - *
5445       }
5446       { \dim_use:N \l_@@_rounded_corners_dim }
5447     }
5448   }
5449   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5450   \dim_compare:nNnT { \l_tmpa_dim } > { \c_zero_dim }
5451   {
5452     \@@_put_in_row_style:e
5453     {
5454       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5455       {

```

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

```

5456         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5457         { \dim_use:N \l_tmpa_dim }
5458       }
5459     }
5460   }

```

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

```

5461   \dim_compare:nNnT { \l_tmpb_dim } > { \c_zero_dim }
5462   {
5463     \@@_put_in_row_style:e
5464     {
5465       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5466       {
5467         \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5468         { \dim_use:N \l_tmpb_dim }
5469       }
5470     }
5471   }

```

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

```

5472   \tl_if_empty:NF \l_@@_color_tl
5473   {
5474     \@@_put_in_row_style:e
5475     {
5476       \mode_leave_vertical:
5477       \@@_color:n { \l_@@_color_tl }

```

```

5478     }
5479 }
\l_@@_bold_row_style_bool is the value of the key bold.
5480 \bool_if:NT \l_@@_bold_row_style_bool
5481 {
5482   \@@_put_in_row_style:n
5483   {
5484     \exp_not:n
5485     {
5486       \if_mode_math:
5487         \c_math_toggle_token
5488         \bfseries \boldmath
5489         \c_math_toggle_token
5490       \else:
5491         \bfseries \boldmath
5492       \fi:
5493     }
5494   }
5495 }
5496 \group_end:
5497 \g_@@_row_style_tl
5498 \ignorespaces
5499 }

```

The following commande must *not* be protected.

```

5500 \cs_new:Npn \@@_rounded_from_row:n #1
5501 {
5502   \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

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

```

5503   { \int_eval:n { #1 } - 1 }
5504   {
5505     \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5506     - \exp_not:n { \int_use:N \c@jCol }
5507   }
5508   { \dim_use:N \l_@@_rounded_corners_dim }
5509 }

```

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).

```
5510 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5511 {
```

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.

```
5512 \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`.

```
5513 \str_if_in:nnF { #1 } { !! }
5514 {
5515 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

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

```
5516 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5517 }
5518 \int_if_zero:nTF { \l_tmpa_int }
```

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

```
5519 {
5520 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5521 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5522 }
```

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

```
5523 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5524 }
5525 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5526 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

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

```
5527 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5528 {
5529 \dim_compare:nNnT { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim }
5530 {
```

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

```
5531 \group_begin:
5532 \pgfsetcornersarced
5533 {
5534 \pgfpoint
5535 { \l_@@_tab_rounded_corners_dim }
5536 { \l_@@_tab_rounded_corners_dim }
5537 }
```

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.

```
5538 \bool_if:NTF \l_@@_hvlines_bool
5539 {
5540 \pgfpathrectanglecorners
5541 {
5542 \pgfpointadd
5543 { \@@_qpoint:n { row-1 } }
5544 { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5545 }
5546 {
5547 \pgfpointadd
5548 {
```

```

5549         \@@_qpoint:n
5550         { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5551     }
5552     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5553 }
5554 }
5555 {
5556     \pgfpathrectanglecorners
5557     { \@@_qpoint:n { row-1 } }
5558     {
5559         \pgfpointadd
5560         {
5561             \@@_qpoint:n
5562             { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5563         }
5564         { \pgfpoint \c_zero_dim \arrayrulewidth }
5565     }
5566 }
5567 \pgfusepath { clip }
5568 \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5569     }
5570 }

```

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`).

```

5571 \cs_new_protected:Npn \@@_actually_color:
5572 {
5573     \pgfpicture
5574     \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.

```

5575     \@@_clip_with_rounded_corners:
5576     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5577     {
5578         \int_compare:nNnTF { ##1 } = { \c_one_int }
5579         {
5580             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5581             \use:c { g_@@_color _ 1 _tl }
5582             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5583         }
5584         {
5585             \begin { pgfscope }
5586                 \@@_color_opacity: ##2
5587                 \use:c { g_@@_color _ ##1 _tl }
5588                 \tl_gclear:c { g_@@_color _ ##1 _tl }
5589                 \pgfusepath { fill }
5590             \end { pgfscope }
5591         }
5592     }
5593 \endpgfpicture
5594 }

```

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

```

5595 \cs_new_protected:Npn \@@_color_opacity:
5596 {
5597     \peek_meaning:NTF [
5598         { \@@_color_opacity:w }
5599         { \@@_color_opacity:w [ ] }
5600     }

```

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.

```

5601 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5602 {
5603   \tl_clear:N \l_tmpa_tl
5604   \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.

```

```

5605   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5606   \tl_if_empty:NTF \l_tmpb_tl
5607     { \@declaredcolor }
5608     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5609 }

```

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

```

5610 \keys_define:nn { nicematrix / color-opacity }
5611 {
5612   opacity .tl_set:N          = \l_tmpa_tl ,
5613   opacity .value_required:n = true
5614 }

```

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

```

5615 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5616 {
5617   \def \l_@@_rows_tl { #1 }
5618   \def \l_@@_cols_tl { #2 }
5619   \@@_cartesian_path:
5620 }

```

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

```

5621 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5622 {
5623   \tl_if_blank:nF { #2 }
5624   {
5625     \@@_add_to_colors_seq:en
5626     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5627     { \@@_cartesian_color:nn { #3 } { - } }
5628   }
5629 }

```

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

```

5630 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5631 {
5632   \tl_if_blank:nF { #2 }
5633   {
5634     \@@_add_to_colors_seq:en
5635     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5636     { \@@_cartesian_color:nn { - } { #3 } }
5637   }
5638 }

```

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

```

5639 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5640 {
5641   \tl_if_blank:nF { #2 }
5642   {
5643     \@@_add_to_colors_seq:en
5644     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5645     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5646   }
5647 }

```


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

```

5648 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5649 {
5650   \tl_if_blank:nF { #2 }
5651   {
5652     \@@_add_to_colors_seq:en
5653     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5654     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5655   }
5656 }

```

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

```

5657 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5658 {
5659   \@@_cut_on_hyphen:w #1 \q_stop
5660   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5661   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5662   \@@_cut_on_hyphen:w #2 \q_stop
5663   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5664   \tl_set:Nx \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`.

```

5665   \@@_cartesian_path:n { #3 }
5666 }

```

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

```

5667 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5668 {
5669   \clist_map_inline:nn { #3 }
5670   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5671 }

```

```

5672 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5673 {
5674   \int_step_inline:nn { \c@iRow }
5675   {
5676     \int_step_inline:nn { \c@jCol }
5677     {
5678       \int_if_even:nTF { #####1 + ##1 }
5679       { \@@_cellcolor [ #1 ] { #2 } }
5680       { \@@_cellcolor [ #1 ] { #3 } }
5681       { ##1 - #####1 }
5682     }
5683   }
5684 }

```

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”.

```

5685 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5686 {
5687   \@@_rectanglecolor [ #1 ] { #2 }
5688   { 1 - 1 }
5689   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5690 }

5691 \keys_define:nn { nicematrix / rowcolors }
5692 {
5693   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,

```

```

5694     respect-blocks .default:n = true ,
5695     cols .tl_set:N = \l_@@_cols_tl ,
5696     restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5697     restart .default:n = true ,
5698     unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5699 }

```

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*.

```

5700 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5701 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5702     \group_begin:
5703     \seq_clear_new:N \l_@@_colors_seq
5704     \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5705     \tl_clear_new:N \l_@@_cols_tl
5706     \tl_set:Nn \l_@@_cols_tl { - }
5707     \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).

```

5708     \int_zero_new:N \l_@@_color_int
5709     \int_set_eq:NN \l_@@_color_int \c_one_int
5710     \bool_if:NT \l_@@_respect_blocks_bool
5711     {

```

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`).

```

5712         \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5713         \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5714         { \@@_not_in_exterior_p:nnnnn ##1 }
5715     }
5716     \pgfpicture
5717     \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5718     \clist_map_inline:nn { #2 }
5719     {
5720         \tl_set:Nn \l_tmpa_tl { ##1 }
5721         \tl_if_in:NnTF \l_tmpa_tl { - }
5722         { \@@_cut_on_hyphen:w ##1 \q_stop }
5723         { \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.

```

5724         \int_set:Nn \l_tmpa_int \l_tmpa_tl
5725         \int_set:Nn \l_@@_color_int
5726         { \bool_if:NNTF \l_@@_rowcolors_restart_bool { 1 } { \l_tmpa_tl } }
5727         \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5728         \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5729         {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5730         \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).

```

5731     \bool_if:NT \l_@@_respect_blocks_bool
5732     {
5733         \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5734         { \@@_intersect_our_row_p:nnnnn ###1 }
5735         \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5736     }
5737     \tl_set:Ne \l_@@_rows_tl
5738     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5739     \tl_set:Ne \l_@@_color_tl
5740     {
5741         \@@_color_index:n
5742         {
5743             \int_mod:nn
5744             { \l_@@_color_int - 1 }
5745             { \seq_count:N \l_@@_colors_seq }
5746             + 1
5747         }
5748     }
5749     \tl_if_empty:NF \l_@@_color_tl
5750     {
5751         \@@_add_to_colors_seq:ee
5752         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5753         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5754     }
5755     \int_incr:N \l_@@_color_int
5756     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5757 }
5758 }
5759 \endpgfpicture
5760 \group_end:
5761 }

```

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.

```

5762 \cs_new:Npn \@@_color_index:n #1
5763 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5764     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5765     { \@@_color_index:n { #1 - 1 } }
5766     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5767 }

```

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.

```

5768 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5769 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around `#3` and `#4` are mandatory.

```

5770 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5771 {
5772     \int_compare:nNnT { #3 } > { \l_tmpb_int }
5773     { \int_set:Nn \l_tmpb_int { #3 } }
5774 }

```

```

5775 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5776 {
5777   \int_if_zero:nTF { #4 }
5778   { \prg_return_false: }
5779   {
5780     \int_compare:nNnTF { #2 } > { \c@jCol }
5781     { \prg_return_false: }
5782     { \prg_return_true: }
5783   }
5784 }

```

The following command return true when the block intersects the row \l_tmpa_int.

```

5785 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5786 {
5787   \int_compare:nNnTF { #1 } > { \l_tmpa_int }
5788   { \prg_return_false: }
5789   {
5790     \int_compare:nNnTF { \l_tmpa_int } > { #3 }
5791     { \prg_return_false: }
5792     { \prg_return_true: }
5793   }
5794 }

```

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).

```

5795 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5796 {
5797   \dim_compare:nNnTF { #1 } = { \c_zero_dim }
5798   {
5799     \bool_if:NTF \l_@@_nocolor_used_bool
5800     { \@@_cartesian_path_normal_ii: }
5801     {
5802       \clist_if_empty:NTF \l_@@_corners_cells_clist
5803       { \@@_cartesian_path_normal_i:n { #1 } }
5804       { \@@_cartesian_path_normal_ii: }
5805     }
5806   }
5807   { \@@_cartesian_path_normal_i:n { #1 } }
5808 }

```

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.

```

5809 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5810 {
5811   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5812   \clist_map_inline:Nn \l_@@_cols_tl
5813   {

```

We use \def instead of \tl_set:Nn for efficiency only.

```

5814     \def \l_tmpa_tl { ##1 }
5815     \tl_if_in:NnTF \l_tmpa_tl { - }
5816     { \@@_cut_on_hyphen:w ##1 \q_stop }
5817     { \def \l_tmpb_tl { ##1 } } % 2025-04-16
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 \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_col_total_int }
5825 { \@@_error:n { Invalid~col~number } }
5826 \tl_if_empty:NTF \l_tmpb_tl
5827 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5828 {
5829     \str_if_eq:eeT \l_tmpb_tl { * }
5830     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5831 }
5832 \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_col_total_int }
5833 { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5834 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5835 \@@_qpoint:n { col - \l_tmpa_tl }
5836 \int_compare:nNnTF { \l_@@_first_col_int } = { \l_tmpa_tl }
5837 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5838 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5839 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5840 \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.

```

5841 \clist_map_inline:Nn \l_@@_rows_tl
5842 {
5843     \def \l_tmpa_tl { #####1 }
5844     \tl_if_in:NnTF \l_tmpa_tl { - }
5845     { \@@_cut_on_hyphen:w #####1 \q_stop }
5846     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5847     \tl_if_empty:NTF \l_tmpa_tl
5848     { \def \l_tmpa_tl { 1 } }
5849     {
5850         \str_if_eq:eeT \l_tmpa_tl { * }
5851         { \def \l_tmpa_tl { 1 } }
5852     }
5853     \tl_if_empty:NTF \l_tmpb_tl
5854     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5855     {
5856         \str_if_eq:eeT \l_tmpb_tl { * }
5857         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5858     }
5859     \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_row_total_int }
5860     { \@@_error:n { Invalid~row~number } }
5861     \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_row_total_int }
5862     { \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`.

```

5863 \cs_if_exist:cF
5864 { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5865 {
5866     \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5867     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5868     \@@_qpoint:n { row - \l_tmpa_tl }
5869     \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5870     \pgfpathrectanglecorners
5871     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5872     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5873 }
5874 }
5875 }
5876 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5877 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5878 {
5879   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5880   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5881   \clist_map_inline:Nn \l_@@_cols_tl
5882   {
5883     \@@_qpoint:n { col - ##1 }
5884     \int_compare:nNnTF { \l_@@_first_col_int } = { ##1 }
5885     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5886     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5887     \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5888     \dim_set:Nn \l_@@_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5889     \clist_map_inline:Nn \l_@@_rows_tl
5890     {
5891       \@@_if_in_corner:nF { #####1 - ##1 }
5892       {
5893         \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5894         \dim_set:Nn \l_@@_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5895         \@@_qpoint:n { row - #####1 }
5896         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5897         \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5898         {
5899           \pgfpathrectanglecorners
5900           { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5901           { \pgfpoint \l_@@_tmpa_dim \l_@@_tmpb_dim }
5902         }
5903       }
5904     }
5905   }
5906 }

```

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`).

```

5907 \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

```

5908 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5909 {
5910   \bool_set_true:N \l_@@_nocolor_used_bool
5911   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5912   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5913   \clist_map_inline:Nn \l_@@_rows_tl
5914   {
5915     \clist_map_inline:Nn \l_@@_cols_tl
5916     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
5917   }
5918 }

```

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.

```

5919 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5920 {
5921   \clist_set_eq:NN \l_@@_tmpa_clist #1

```

```

5922 \clist_clear:N #1
5923 \clist_map_inline:Nn \l_tmpa_clist
5924 {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

5925 \def \l_tmpa_tl { ##1 }
5926 \tl_if_in:NnTF \l_tmpa_tl { - }
5927 { \@@_cut_on_hyphen:w ##1 \q_stop }
5928 { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5929 \bool_lazy_or:nnT
5930 { \str_if_eq_p:ee \l_tmpa_tl { * } }
5931 { \tl_if_blank_p:o \l_tmpa_tl }
5932 { \def \l_tmpa_tl { 1 } }
5933 \bool_lazy_or:nnT
5934 { \str_if_eq_p:ee \l_tmpb_tl { * } }
5935 { \tl_if_blank_p:o \l_tmpb_tl }
5936 { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5937 \int_compare:nNnT { \l_tmpb_tl } > { #2 }
5938 { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5939 \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
5940 { \clist_put_right:Nn #1 { #####1 } }
5941 }
5942 }

```

The following command will be linked to `\cellcolor` in the tabular.

```

5943 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5944 {
5945 \tl_gput_right:Ne \g_@@_pre_code_before_tl
5946 {

```

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`).

```

5947 \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5948 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5949 }
5950 \ignorespaces
5951 }

```

The following command will be linked to `\rowcolor` in the tabular.

```

5952 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5953 {
5954 \tl_gput_right:Ne \g_@@_pre_code_before_tl
5955 {
5956 \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5957 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5958 { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5959 }
5960 \ignorespaces
5961 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5962 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5963 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around `#2` and `#3` are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

5964 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5965 {

```

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`.

```

5966 \seq_gclear:N \g_tmpa_seq
5967 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5968 { \@@_rowlistcolors_tabular:nnnn #1 }
5969 \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`.

```

5970 \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5971 {
5972   { \int_use:N \c@iRow }
5973   { \exp_not:n { #1 } }
5974   { \exp_not:n { #2 } }
5975   { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5976 }
5977 \ignorespaces
5978 }

```

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`).

```

5979 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
5980 {
5981   \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).

```

5982 { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5983 {
5984   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5985   {
5986     \@@_rowlistcolors
5987     [ \exp_not:n { #2 } ]
5988     { #1 - \int_eval:n { \c@iRow - 1 } }
5989     { \exp_not:n { #3 } }
5990     [ \exp_not:n { #4 } ]
5991   }
5992 }
5993 }

```

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.

```

5994 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5995 {
5996   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5997   { \@@_rowlistcolors_tabular_ii:nnnn #1 }
5998   \seq_gclear:N \g_@@_rowlistcolors_seq
5999 }

```



```

6000 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6001 {
6002   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6003   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6004 }

```

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.

```

6005 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
6006 {

```

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).

```

6007   \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
6008   {

```

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).

```

6009     \tl_gput_left:Ne \g_@@_pre_code_before_tl
6010     {
6011       \exp_not:N \columncolor [ #1 ]
6012       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6013     }
6014   }
6015 }

```

```

6016 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6017 {
6018   \clist_map_inline:nn { #1 }
6019   {
6020     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6021     { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6022     \columncolor { nocolor } { ##1 }
6023   }
6024 }

```

```

6025 \cs_new_protected:Npn \@@_EmptyRow:n #1
6026 {
6027   \clist_map_inline:nn { #1 }
6028   {
6029     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6030     { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6031     \rowcolor { nocolor } { ##1 }
6032   }
6033 }

```

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`.

```
6034 \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`.

```
6035 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6036 {
6037   \int_if_zero:nTF { \l_@@_first_col_int }
6038   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6039   {
6040     \int_if_zero:nTF { \c_jCol }
6041     {
6042       \int_compare:nNnF { \c_iRow } = { -1 }
6043       {
6044         \int_compare:nNnF { \c_iRow } = { \l_@@_last_row_int - 1 }
6045         { #1 }
6046       }
6047     }
6048     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6049   }
6050 }
```

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.

```
6051 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6052 {
6053   \int_if_zero:nF { \c_iRow }
6054   {
6055     \int_compare:nNnF { \c_iRow } = { \l_@@_last_row_int }
6056     {
6057       \int_compare:nNnT { \c_jCol } > { \c_zero_int }
6058       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6059     }
6060   }
6061 }
```

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`.

```
6062 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6063 {
6064   \IfPackageLoadedTF { tikz }
6065   {
6066     \IfPackageLoadedTF { booktabs }
6067     { #2 }
6068     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6069   }
6070   { \@@_error:nn { TopRule~without~tikz } { #1 } }
6071 }

6072 \NewExpandableDocumentCommand { \@@_TopRule } { }
6073 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }

6074 \cs_new:Npn \@@_TopRule_i:
6075 {
6076   \noalign \bgroup
6077   \peek_meaning:NTF [
6078   { \@@_TopRule_ii: }
```

```

6079         { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6080     }
6081 \NewDocumentCommand \@@_TopRule_ii: { o }
6082 {
6083     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6084     {
6085         \@@_hline:n
6086         {
6087             position = \int_eval:n { \c@iRow + 1 } ,
6088             tikz =
6089             {
6090                 line-width = #1 ,
6091                 yshift = 0.25 \arrayrulewidth ,
6092                 shorten~< = - 0.5 \arrayrulewidth
6093             } ,
6094             total-width = #1
6095         }
6096     }
6097     \skip_vertical:n { \belowrulesep + #1 }
6098     \egroup
6099 }
6100 \NewExpandableDocumentCommand { \@@_BottomRule } { } { }
6101 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }
6102 \cs_new:Npn \@@_BottomRule_i:
6103 {
6104     \noalign \bgroup
6105     \peek_meaning:NTF [
6106         { \@@_BottomRule_ii: }
6107         { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6108     }
6109 \NewDocumentCommand \@@_BottomRule_ii: { o }
6110 {
6111     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6112     {
6113         \@@_hline:n
6114         {
6115             position = \int_eval:n { \c@iRow + 1 } ,
6116             tikz =
6117             {
6118                 line-width = #1 ,
6119                 yshift = 0.25 \arrayrulewidth ,
6120                 shorten~< = - 0.5 \arrayrulewidth
6121             } ,
6122             total-width = #1 ,
6123         }
6124     }
6125     \skip_vertical:N \aboverulesep
6126     \@@_create_row_node_i:
6127     \skip_vertical:n { #1 }
6128     \egroup
6129 }
6130 \NewExpandableDocumentCommand { \@@_MidRule } { } { }
6131 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6132 \cs_new:Npn \@@_MidRule_i:
6133 {
6134     \noalign \bgroup
6135     \peek_meaning:NTF [
6136         { \@@_MidRule_ii: }
6137         { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6138     }
6139 \NewDocumentCommand \@@_MidRule_ii: { o }

```

```

6140 {
6141   \skip_vertical:N \aboverulesep
6142   \@@_create_row_node_i:
6143   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6144   {
6145     \@@_hline:n
6146     {
6147       position = \int_eval:n { \c@iRow + 1 } ,
6148       tikz =
6149       {
6150         line-width = #1 ,
6151         yshift = 0.25 \arrayrulewidth ,
6152         shorten~< = - 0.5 \arrayrulewidth
6153       } ,
6154       total-width = #1 ,
6155     }
6156   }
6157   \skip_vertical:n { \belowrulesep + #1 }
6158   \egroup
6159 }

```

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.

```

6160 \keys_define:nn { nicematrix / Rules }
6161 {
6162   position .int_set:N = \l_@@_position_int ,
6163   position .value_required:n = true ,
6164   start .int_set:N = \l_@@_start_int ,
6165   end .code:n =
6166     \bool_lazy_or:nnTF
6167     { \tl_if_empty_p:n { #1 } }
6168     { \str_if_eq_p:ee { #1 } { last } }
6169     { \int_set_eq:NN \l_@@_end_int \c@jCol }
6170     { \int_set:Nn \l_@@_end_int { #1 } }
6171 }

```

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.

```

6172 \keys_define:nn { nicematrix / RulesBis }
6173 {
6174   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6175   multiplicity .initial:n = 1 ,
6176   dotted .bool_set:N = \l_@@_dotted_bool ,
6177   dotted .initial:n = false ,
6178   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]`.

```

6179   color .code:n =
6180     \@@_set_CTarc:n { #1 }

```

```

6181     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6182     color .value_required:n = true ,
6183     sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6184     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.

```

6185     tikz .code:n =
6186     \IfPackageLoadedTF { tikz }
6187     { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6188     { \@@_error:n { tikz-without-tikz } } ,
6189     tikz .value_required:n = true ,
6190     total-width .dim_set:N = \l_@@_rule_width_dim ,
6191     total-width .value_required:n = true ,
6192     width .meta:n = { total-width = #1 } ,
6193     unknown .code:n = \@@_error:n { Unknown~key~for~RulesBis }
6194 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6195 \cs_new_protected:Npn \@@_vline:n #1
6196 {

```

The group is for the options.

```

6197     \group_begin:
6198     \int_set_eq:NN \l_@@_end_int \c@iRow
6199     \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).

```

6200     \int_compare:nNnT { \l_@@_position_int } < { \c@jCol + 2 }
6201     \@@_vline_i:
6202     \group_end:
6203 }
6204 \cs_new_protected:Npn \@@_vline_i:
6205 {

```

`\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`.

```

6206     \tl_set:Nn \l_tmpb_tl { \int_use:N \l_@@_position_int }
6207     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6208     \l_tmpa_tl
6209     {

```

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.

```

6210     \bool_gset_true:N \g_tmpa_bool
6211     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6212     { \@@_test_vline_in_block:nnnnn ##1 }
6213     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6214     { \@@_test_vline_in_block:nnnnn ##1 }
6215     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6216     { \@@_test_vline_in_stroken_block:nnnn ##1 }
6217     \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_v: }
6218     \bool_if:NTF \g_tmpa_bool
6219     {
6220         \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.

```

6221         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6222     }
6223     {
6224         \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6225         {
6226             \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6227             \@@_vline_ii:
6228             \int_zero:N \l_@@_local_start_int
6229         }
6230     }
6231 }
6232 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6233 {
6234     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6235     \@@_vline_ii:
6236 }
6237 }

6238 \cs_new_protected:Npn \@@_test_in_corner_v:
6239 {
6240     \int_compare:nNnTF { \l_tmpb_tl } = { \c_jCol + 1 }
6241     {
6242         \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6243         { \bool_set_false:N \g_tmpa_bool }
6244     }
6245     {
6246         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6247         {
6248             \int_compare:nNnTF { \l_tmpb_tl } = { \c_one_int }
6249             { \bool_set_false:N \g_tmpa_bool }
6250             {
6251                 \@@_if_in_corner:nT
6252                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6253                 { \bool_set_false:N \g_tmpa_bool }
6254             }
6255         }
6256     }
6257 }

6258 \cs_new_protected:Npn \@@_vline_ii:
6259 {
6260     \tl_clear:N \l_@@_tikz_rule_tl
6261     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6262     \bool_if:NTF \l_@@_dotted_bool
6263     { \@@_vline_iv: }
6264     {
6265         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6266         { \@@_vline_iii: }
6267         { \@@_vline_v: }
6268     }
6269 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

6270 \cs_new_protected:Npn \@@_vline_iii:
6271 {
6272     \pgfpicture
6273     \pgfrememberpicturepositiononpagetrue
6274     \pgf@relevantforpicturesizefalse
6275     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }

```

```

6276 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6277 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6278 \dim_set:Nn \l_tmpb_dim
6279 {
6280   \pgf@x
6281   - 0.5 \l_@@_rule_width_dim
6282   +
6283   ( \arrayrulewidth * \l_@@_multiplicity_int
6284     + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6285 }
6286 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6287 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6288 \bool_lazy_all:nT
6289 {
6290   { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6291   { \cs_if_exist_p:N \CT@drsc@ }
6292   { ! \tl_if_blank_p:o \CT@drsc@ }
6293 }
6294 {
6295   \group_begin:
6296   \CT@drsc@
6297   \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6298   \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6299   \dim_set:Nn \l_@@_tmpd_dim
6300   {
6301     \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6302     * ( \l_@@_multiplicity_int - 1 )
6303   }
6304   \pgfpathrectanglecorners
6305   { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6306   { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6307   \pgfusepath { fill }
6308   \group_end:
6309 }
6310 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6311 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6312 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6313 {
6314   \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6315   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6316   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6317 }
6318 \CT@arc@
6319 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6320 \pgfsetrectcap
6321 \pgfusepathqstroke
6322 \endpgfpicture
6323 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6324 \cs_new_protected:Npn \@@_vline_iv:
6325 {
6326   \pgfpicture
6327   \pgfrememberpicturepositiononpagetrue
6328   \pgf@relevantforpicturesizefalse
6329   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6330   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6331   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6332   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6333   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6334   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6335   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6336   \CT@arc@

```

```

6337 \@@_draw_line:
6338 \endpgfpicture
6339 }

```

The following code is for the case when the user uses the key `tikz`.

```

6340 \cs_new_protected:Npn \@@_vline_v:
6341 {
6342   \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).

```

6343   \CT@arc@
6344   \tl_if_empty:NF \l_@@_rule_color_tl
6345   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6346   \pgfrememberpicturepositiononpagetrue
6347   \pgf@relevantforpicturesizefalse
6348   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6349   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6350   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6351   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6352   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6353   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6354   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6355   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6356   ( \l_tmpb_dim , \l_tmpa_dim ) --
6357   ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6358   \end { tikzpicture }
6359 }

```

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).

```

6360 \cs_new_protected:Npn \@@_draw_vlines:
6361 {
6362   \int_step_inline:nnn
6363   {
6364     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6365     { 2 }
6366     { 1 }
6367   }
6368   {
6369     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6370     { \c@jCol }
6371     { \int_eval:n { \c@jCol + 1 } }
6372   }
6373   {
6374     \str_if_eq:eeF \l_@@_vlines_clist { all }
6375     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6376     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6377   }
6378 }

```

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}`.

```

6379 \cs_new_protected:Npn \@@_hline:n #1
6380 {

```


The group is for the options.

```

6381 \group_begin:
6382 \int_set_eq:NN \l_@@_end_int \c@jCol
6383 \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6384 \@@_hline_i:
6385 \group_end:
6386 }

6387 \cs_new_protected:Npn \@@_hline_i:
6388 {
6389 % \int_zero:N \l_@@_local_start_int
6390 % \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 $\l_@@_tmpc_tl$.

```

6391 \tl_set:No \l\_tmpa\_tl { \int_use:N \l_@@_position_int }
6392 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6393 \l\_tmpb\_tl
6394 {

```

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 $\backslash\text{Block}$ or a virtual block corresponding to a dotted line, created by $\backslash\text{Cdots}$, $\backslash\text{Vdots}$, etc.), we will set \g_tmpa_bool to false and the small horizontal rule won't be drawn.

```

6395 \bool_gset_true:N \g\_tmpa\_bool

```

We test whether we are in a block.

```

6396 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6397 { \@@_test_hline_in_block:nnnnn ##1 }

6398 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6399 { \@@_test_hline_in_block:nnnnn ##1 }

6400 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6401 { \@@_test_hline_in_stroken_block:nnnn ##1 }

6402 \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_h: }
6403 \bool_if:NTF \g\_tmpa\_bool
6404 {
6405 \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.

```

6406 { \int_set:Nn \l_@@_local_start_int \l\_tmpb\_tl }
6407 }
6408 {
6409 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6410 {
6411 \int_set:Nn \l_@@_local_end_int { \l\_tmpb\_tl - 1 }
6412 \@@_hline_ii:
6413 \int_zero:N \l_@@_local_start_int
6414 }
6415 }
6416 }
6417 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6418 {
6419 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6420 \@@_hline_ii:
6421 }
6422 }

```

```

6423 \cs_new_protected:Npn \@@_test_in_corner_h:
6424 {
6425 \int_compare:nNnTF { \l\_tmpa\_tl } = { \c@iRow + 1 }
6426 {
6427 \@@_if_in_corner:nT { \int_eval:n { \l\_tmpa\_tl - 1 } - \l\_tmpb\_tl }

```

```

6428         { \bool_set_false:N \g_tmpa_bool }
6429     }
6430     {
6431         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6432         {
6433             \int_compare:nNnTF { \l_tmpa_tl } = { \c_one_int }
6434             { \bool_set_false:N \g_tmpa_bool }
6435             {
6436                 \@@_if_in_corner:nT
6437                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6438                 { \bool_set_false:N \g_tmpa_bool }
6439             }
6440         }
6441     }
6442 }

```

```

6443 \cs_new_protected:Npn \@@_hline_ii:
6444 {
6445     \tl_clear:N \l_@@_tikz_rule_tl
6446     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6447     \bool_if:NTF \l_@@_dotted_bool
6448     { \@@_hline_iv: }
6449     {
6450         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6451         { \@@_hline_iii: }
6452         { \@@_hline_v: }
6453     }
6454 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6455 \cs_new_protected:Npn \@@_hline_iii:
6456 {
6457     \pgfpicture
6458     \pgfrememberpicturepositiononpagetrue
6459     \pgf@relevantforpicturesizefalse
6460     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6461     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6462     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6463     \dim_set:Nn \l_tmpb_dim
6464     {
6465         \pgf@y
6466         - 0.5 \l_@@_rule_width_dim
6467         +
6468         ( \arrayrulewidth * \l_@@_multiplicity_int
6469           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6470     }
6471     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6472     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6473     \bool_lazy_all:nT
6474     {
6475         { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6476         { \cs_if_exist_p:N \CT@drsc@ }
6477         { ! \tl_if_blank_p:o \CT@drsc@ }
6478     }
6479     {
6480         \group_begin:
6481         \CT@drsc@
6482         \dim_set:Nn \l_@@_tmpd_dim
6483         {
6484             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6485             * ( \l_@@_multiplicity_int - 1 )
6486         }

```

```

6487 \pgfpathrectanglecorners
6488 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6489 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6490 \pgfusepathqfill
6491 \group_end:
6492 }
6493 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6494 \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6495 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6496 {
6497 \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6498 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6499 \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6500 }
6501 \CT@arc@
6502 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6503 \pgfsetrectcap
6504 \pgfusepathqstroke
6505 \endpgfpicture
6506 }

```

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \dots & \dots & \dots & \dots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6507 \cs_new_protected:Npn \l_@@_hline_iv:
6508 {
6509 \pgfpicture
6510 \pgfrememberpicturepositiononpagetrue
6511 \pgf@relevantforpicturesizefalse
6512 \l_@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6513 \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6514 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6515 \l_@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6516 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6517 \int_compare:nNnT { \l_@@_local_start_int } = { \c_one_int }
6518 {
6519 \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6520 \bool_if:NF \g_@@_delims_bool
6521 { \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.

```

6522 \tl_if_eq:NnF \g_@@_left_delim_tl (
6523 { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6524 )
6525 \l_@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6526 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6527 \int_compare:nNnT { \l_@@_local_end_int } = { \c_jCol }

```

```

6528 {
6529   \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6530   \bool_if:NF \g_@@_delims_bool
6531   { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6532   \tl_if_eq:NnF \g_@@_right_delim_tl )
6533   { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6534 }
6535 \CT@arc@
6536 \@@_draw_line:
6537 \endpgfpicture
6538 }

```

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`).

```

6539 \cs_new_protected:Npn \@@_hline_v:
6540 {
6541   \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).

```

6542   \CT@arc@
6543   \tl_if_empty:NF \l_@@_rule_color_tl
6544   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6545   \pgfrememberpicturepositiononpagetrue
6546   \pgf@relevantforpicturesizefalse
6547   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6548   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6549   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6550   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6551   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6552   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6553   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6554   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6555   ( \l_tmpa_dim , \l_tmpb_dim ) --
6556   ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6557   \end { tikzpicture }
6558 }

```

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).

```

6559 \cs_new_protected:Npn \@@_draw_hlines:
6560 {
6561   \int_step_inline:nnn
6562   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6563   {
6564     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6565     { \c@iRow }
6566     { \int_eval:n { \c@iRow + 1 } }
6567   }
6568   {
6569     \str_if_eq:eeF \l_@@_hlines_clist { all }
6570     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6571     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6572   }
6573 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6574 \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.

```

6575 \cs_set:Npn \@@_Hline_i:n #1
6576 {
6577   \peek_remove_spaces:n
6578   {
6579     \peek_meaning:NTF \Hline
6580     { \@@_Hline_ii:nn { #1 + 1 } }
6581     { \@@_Hline_iii:n { #1 } }
6582   }
6583 }
6584 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6585 \cs_set:Npn \@@_Hline_iii:n #1
6586 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6587 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6588 {
6589   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6590   \skip_vertical:N \l_@@_rule_width_dim
6591   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6592   {
6593     \@@_hline:n
6594     {
6595       multiplicity = #1 ,
6596       position = \int_eval:n { \c@iRow + 1 } ,
6597       total-width = \dim_use:N \l_@@_rule_width_dim ,
6598       #2
6599     }
6600   }
6601   \egroup
6602 }

```

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`).

```

6603 \cs_new_protected:Npn \@@_custom_line:n #1
6604 {
6605   \str_clear_new:N \l_@@_command_str
6606   \str_clear_new:N \l_@@_ccommand_str
6607   \str_clear_new:N \l_@@_letter_str
6608   \tl_clear_new:N \l_@@_other_keys_tl
6609   \keys_set:known { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6610   \bool_lazy_all:nTF
6611   {
6612     { \str_if_empty_p:N \l_@@_letter_str }
6613     { \str_if_empty_p:N \l_@@_command_str }
6614     { \str_if_empty_p:N \l_@@_ccommand_str }
6615   }
6616   { \@@_error:n { No~letter~and~no~command } }
6617   { \@@_custom_line_i:o \l_@@_other_keys_tl }
6618 }
6619 \keys_define:nn { nicematrix / custom-line }
6620 {
6621   letter .str_set:N = \l_@@_letter_str ,

```

```

6622     letter .value_required:n = true ,
6623     command .str_set:N = \l_@@_command_str ,
6624     command .value_required:n = true ,
6625     ccommand .str_set:N = \l_@@_ccommand_str ,
6626     ccommand .value_required:n = true ,
6627 }

```

```

6628 \cs_new_protected:Npn \@@_custom_line_i:n #1
6629 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6630     \bool_set_false:N \l_@@_tikz_rule_bool
6631     \bool_set_false:N \l_@@_dotted_rule_bool
6632     \bool_set_false:N \l_@@_color_bool
6633     \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6634     \bool_if:NT \l_@@_tikz_rule_bool
6635     {
6636         \IfPackageLoadedF { tikz }
6637         { \@@_error:n { tikz~in~custom~line~without~tikz } }
6638         \bool_if:NT \l_@@_color_bool
6639         { \@@_error:n { color~in~custom~line~with~tikz } }
6640     }
6641     \bool_if:NT \l_@@_dotted_rule_bool
6642     {
6643         \int_compare:nNnT { \l_@@_multiplicity_int } > { \c_one_int }
6644         { \@@_error:n { key~multiplicity~with~dotted } }
6645     }
6646     \str_if_empty:NF \l_@@_letter_str
6647     {
6648         \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6649         { \@@_error:n { Several~letters } }
6650         {
6651             \tl_if_in:NoTF
6652             \c_@@_forbidden_letters_str
6653             \l_@@_letter_str
6654             { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6655         }

```

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.

```

6656         \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6657         { \@@_v_custom_line:n { #1 } }
6658     }
6659 }
6660 }
6661 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6662 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6663 }
6664 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6665 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6666 \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.

```

6667 \keys_define:nn { nicematrix / custom-line-bis }
6668 {
6669     multiplicity .int_set:N = \l_@@_multiplicity_int ,

```

```

6670     multiplicity .initial:n = 1 ,
6671     multiplicity .value_required:n = true ,
6672     color .code:n = \bool_set_true:N \l_@@_color_bool ,
6673     color .value_required:n = true ,
6674     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6675     tikz .value_required:n = true ,
6676     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6677     dotted .value_forbidden:n = true ,
6678     total-width .code:n = { } ,
6679     total-width .value_required:n = true ,
6680     width .code:n = { } ,
6681     width .value_required:n = true ,
6682     sep-color .code:n = { } ,
6683     sep-color .value_required:n = true ,
6684     unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
6685 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6686 \bool_new:N \l_@@_dotted_rule_bool
6687 \bool_new:N \l_@@_tikz_rule_bool
6688 \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`.

```

6689 \keys_define:nn { nicematrix / custom-line-width }
6690 {
6691     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6692     multiplicity .initial:n = 1 ,
6693     multiplicity .value_required:n = true ,
6694     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6695     total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6696                     \bool_set_true:N \l_@@_total_width_bool ,
6697     total-width .value_required:n = true ,
6698     width .meta:n = { total-width = #1 } ,
6699     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6700 }

```

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`).

```

6701 \cs_new_protected:Npn \@@_h_custom_line:n #1
6702 {

```

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`).

```

6703     \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6704     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6705 }

```

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`).

```

6706 \cs_new_protected:Npn \@@_c_custom_line:n #1
6707 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

6708     \exp_args:Nc \NewExpandableDocumentCommand
6709     { nicematrix - \l_@@_ccommand_str }
6710     { 0 { } m }

```

```

6711 {
6712   \noalign
6713   {
6714     \@@_compute_rule_width:n { #1 , ##1 }
6715     \skip_vertical:n { \l_@@_rule_width_dim }
6716     \clist_map_inline:nn
6717       { ##2 }
6718       { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6719   }
6720 }
6721 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6722 }

```

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*.

```

6723 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6724 {
6725   \tl_if_in:nnTF { #2 } { - }
6726     { \@@_cut_on_hyphen:w #2 \q_stop }
6727     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6728   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6729   {
6730     \@@_hline:n
6731     {
6732       #1 ,
6733       start = \l_tmpa_tl ,
6734       end = \l_tmpb_tl ,
6735       position = \int_eval:n { \c@iRow + 1 } ,
6736       total-width = \dim_use:N \l_@@_rule_width_dim
6737     }
6738   }
6739 }
6740 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6741 {
6742   \bool_set_false:N \l_@@_tikz_rule_bool
6743   \bool_set_false:N \l_@@_total_width_bool
6744   \bool_set_false:N \l_@@_dotted_rule_bool
6745   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6746   \bool_if:NF \l_@@_total_width_bool
6747   {
6748     \bool_if:NTF \l_@@_dotted_rule_bool
6749       { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6750       {
6751         \bool_if:NF \l_@@_tikz_rule_bool
6752         {
6753           \dim_set:Nn \l_@@_rule_width_dim
6754             {
6755               \arrayrulewidth * \l_@@_multiplicity_int
6756               + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6757             }
6758         }
6759       }
6760   }
6761 }
6762 \cs_new_protected:Npn \@@_v_custom_line:n #1
6763 {
6764   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6765   \tl_gput_right:Ne \g_@@_array_preamble_tl
6766     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6767   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6768   {

```



```

6769 \@@_vline:n
6770 {
6771     #1 ,
6772     position = \int_eval:n { \c@jCol + 1 } ,
6773     total-width = \dim_use:N \l_@@_rule_width_dim
6774 }
6775 }
6776 \@@_rec_preamble:n
6777 }
6778 \@@_custom_line:n
6779 { 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.

```

6780 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6781 {
6782     \int_compare:nNnT { \l_tmpa_tl } > { #1 }
6783     {
6784         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6785         {
6786             \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6787             {
6788                 \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6789                 { \bool_gset_false:N \g_tmpa_bool }
6790             }
6791         }
6792     }
6793 }

```

The same for vertical rules.

```

6794 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6795 {
6796     \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6797     {
6798         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6799         {
6800             \int_compare:nNnT { \l_tmpb_tl } > { #2 }
6801             {
6802                 \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6803                 { \bool_gset_false:N \g_tmpa_bool }
6804             }
6805         }
6806     }
6807 }
6808 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6809 {
6810     \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6811     {
6812         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6813         {
6814             \int_compare:nNnTF { \l_tmpa_tl } = { #1 }
6815             { \bool_gset_false:N \g_tmpa_bool }
6816             {
6817                 \int_compare:nNnT { \l_tmpa_tl } = { #3 + 1 }
6818                 { \bool_gset_false:N \g_tmpa_bool }
6819             }
6820         }
6821     }
6822 }

```

```

6823 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6824 {
6825   \int_compare:nNtT { \l_tmpa_tl } > { #1 - 1 }
6826   {
6827     \int_compare:nNtT { \l_tmpa_tl } < { #3 + 1 }
6828     {
6829       \int_compare:nNtTF { \l_tmpb_tl } = { #2 }
6830       { \bool_gset_false:N \g_tmpa_bool }
6831       {
6832         \int_compare:nNtT { \l_tmpb_tl } = { #4 + 1 }
6833         { \bool_gset_false:N \g_tmpa_bool }
6834       }
6835     }
6836   }
6837 }

```

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.

```

6838 \cs_new_protected:Npn \@@_compute_corners:
6839 {
6840   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6841   { \@@_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`).

```

6842   \clist_clear:N \l_@@_corners_cells_clist
6843   \clist_map_inline:Nn \l_@@_corners_cells_clist
6844   {
6845     \str_case:nnF { ##1 }
6846     {
6847       { NW }
6848       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6849       { NE }
6850       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6851       { SW }
6852       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6853       { SE }
6854       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6855     }
6856     { \@@_error:nn { bad~corner } { ##1 } }
6857   }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6858   \clist_if_empty:NF \l_@@_corners_cells_clist
6859   {

```

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.

```

6860     \tl_gput_right:Ne \g_@@_aux_tl
6861     {
6862       \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
6863       { \l_@@_corners_cells_clist }
6864     }
6865   }
6866 }

```

```

6867 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6868 {
6869   \int_step_inline:nnn { #1 } { #3 }
6870   {
6871     \int_step_inline:nnn { #2 } { #4 }
6872     { \cs_set_nopar:cpn { @@ _ block _ ##1 - ###1 } { } }
6873   }
6874 }

6875 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6876 {
6877   \cs_if_exist:cTF
6878   { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6879   { \prg_return_true: }
6880   { \prg_return_false: }
6881 }

```

“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.

```

6882 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6883 {

```

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.

```

6884   \bool_set_false:N \l_tmpa_bool
6885   \int_zero_new:N \l_@@_last_empty_row_int
6886   \int_set:Nn \l_@@_last_empty_row_int { #1 }
6887   \int_step_inline:nnnn { #1 } { #3 } { #5 }
6888   {
6889     \bool_lazy_or:nnTF
6890     {
6891       \cs_if_exist_p:c
6892       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6893     }
6894     { \@@_if_in_block_p:nn { ##1 } { #2 } }
6895     { \bool_set_true:N \l_tmpa_bool }
6896   }
6897   \bool_if:NF \l_tmpa_bool
6898   { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6899 }
6900 }

```

Now, you determine the last empty cell in the row of number 1.

```

6901   \bool_set_false:N \l_tmpa_bool
6902   \int_zero_new:N \l_@@_last_empty_column_int
6903   \int_set:Nn \l_@@_last_empty_column_int { #2 }
6904   \int_step_inline:nnnn { #2 } { #4 } { #6 }
6905   {
6906     \bool_lazy_or:nnTF
6907     {

```

```

6908     \cs_if_exist_p:c
6909     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6910   }
6911   { \@@_if_in_block_p:nn { #1 } { ##1 } }
6912   { \bool_set_true:N \l_tmpa_bool }
6913   {
6914     \bool_if:NF \l_tmpa_bool
6915     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6916   }
6917 }

```

Now, we loop over the rows.

```

6918   \int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int }
6919   {

```

We treat the row number ##1 with another loop.

```

6920     \bool_set_false:N \l_tmpa_bool
6921     \int_step_inline:nnnn { #2 } { #4 } { \l_@@_last_empty_column_int }
6922     {
6923       \bool_lazy_or:nnTF
6924       { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6925       { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6926       { \bool_set_true:N \l_tmpa_bool }
6927       {
6928         \bool_if:NF \l_tmpa_bool
6929         {
6930           \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6931           \clist_put_right:Nn
6932           \l_@@_corners_cells_clist
6933           { ##1 - #####1 }
6934           \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6935         }
6936       }
6937     }
6938   }
6939 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6940 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6941 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }

```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient:
`\clist_if_in:NcT \l_@@_corners_cells_clist { #1 } ...`

24 The environment {NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```

6942 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

6943 \keys_define:nn { nicematrix / NiceMatrixBlock }
6944 {
6945   auto-columns-width .code:n =
6946   {
6947     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6948     \dim_gzero_new:N \g_@@_max_cell_width_dim
6949     \bool_set_true:N \l_@@_auto_columns_width_bool
6950   }
6951 }

```

```

6952 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6953 {
6954   \int_gincr:N \g_@@_NiceMatrixBlock_int
6955   \dim_zero:N \l_@@_columns_width_dim
6956   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6957   \bool_if:NT \l_@@_block_auto_columns_width_bool
6958   {
6959     \cs_if_exist:cT
6960     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6961     {
6962       \dim_set:Nn \l_@@_columns_width_dim
6963       {
6964         \use:c
6965         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6966       }
6967     }
6968   }
6969 }

```

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`).

```

6970 {
6971   \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.

```

6972   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6973   {
6974     \bool_if:NT \l_@@_block_auto_columns_width_bool
6975     {
6976       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6977       \iow_shipout:Ne \@mainaux
6978       {
6979         \cs_gset:cpn
6980         { @@ _ 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.

```

6981         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6982       }
6983       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6984     }
6985   }
6986   \ignorespacesafterend
6987 }

```

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).

```

6988 \cs_new_protected:Npn \@@_create_extra_nodes:
6989 {
6990   \bool_if:nTF \l_@@_medium_nodes_bool
6991   {
6992     \bool_if:NTF \l_@@_no_cell_nodes_bool
6993     { \@@_error:n { extra-nodes~with~no~cell~nodes } }
6994     {
6995       \bool_if:NTF \l_@@_large_nodes_bool

```

```

6996         \@@_create_medium_and_large_nodes:
6997         \@@_create_medium_nodes:
6998     }
6999 }
7000 {
7001     \bool_if:NT \l_@@_large_nodes_bool
7002     {
7003         \bool_if:NTF \l_@@_no_cell_nodes_bool
7004         { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7005         \@@_create_large_nodes:
7006     }
7007 }
7008 }

```

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`.

```

7009 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7010 {
7011     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7012     {
7013         \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7014         \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7015         \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7016         \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7017     }
7018     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7019     {
7020         \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7021         \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7022         \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7023         \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7024     }

```

We begin the two nested loops over the rows and the columns of the array.

```

7025     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7026     {
7027         \int_step_variable:nnNn
7028         \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.

```

7029     {
7030         \cs_if_exist:cT
7031         { 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`.

```

7032     {
7033         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7034         \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7035         { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7036         \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7037         {
7038             \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7039             { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7040         }

```

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`.

```

7041         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7042         \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7043         { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } { \pgf@y } }
7044         \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7045         {
7046             \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7047             { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } { \pgf@x } }
7048         }
7049     }
7050 }
7051 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7052 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7053 {
7054     \dim_compare:nNnT
7055     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7056     {
7057         \@@_qpoint:n { row - \@@_i: - base }
7058         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7059         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7060     }
7061 }
7062 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7063 {
7064     \dim_compare:nNnT
7065     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7066     {
7067         \@@_qpoint:n { col - \@@_j: }
7068         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7069         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7070     }
7071 }
7072 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7073 \cs_new_protected:Npn \@@_create_medium_nodes:
7074 {
7075     \pgfpicture
7076     \pgfrememberpicturepositiononpagetrue
7077     \pgf@relevantforpicturesizefalse
7078     \@@_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”.

```

7079     \tl_set:Nn \l_@@_suffix_tl { -medium }
7080     \@@_create_nodes:

```

```

7081 \endpgfpicture
7082 }

```

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:`.

```

7083 \cs_new_protected:Npn \@@_create_large_nodes:
7084 {
7085   \pgfpicture
7086   \pgfrememberpicturepositiononpagetrue
7087   \pgf@relevantforpicturesizefalse
7088   \@@_computations_for_medium_nodes:
7089   \@@_computations_for_large_nodes:
7090   \tl_set:Nn \l_@@_suffix_tl { - large }
7091   \@@_create_nodes:
7092   \endpgfpicture
7093 }
7094 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7095 {
7096   \pgfpicture
7097   \pgfrememberpicturepositiononpagetrue
7098   \pgf@relevantforpicturesizefalse
7099   \@@_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”.

```

7100   \tl_set:Nn \l_@@_suffix_tl { - medium }
7101   \@@_create_nodes:
7102   \@@_computations_for_large_nodes:
7103   \tl_set:Nn \l_@@_suffix_tl { - large }
7104   \@@_create_nodes:
7105   \endpgfpicture
7106 }

```

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.

```

7107 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7108 {
7109   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7110   \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`.

```

7111   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7112   {
7113     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7114     {
7115       (
7116         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7117         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7118       )
7119       / 2
7120     }
7121     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7122     { l_@@_row _ \@@_i: _ min _ dim }
7123   }
7124   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:

```

¹⁵If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`


```

7125 {
7126   \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7127   {
7128     (
7129       \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7130       \dim_use:c
7131         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7132     )
7133     / 2
7134   }
7135   \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7136   { l_@@_column _ \@@_j: _ max _ dim }
7137 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7138 \dim_sub:cn
7139   { l_@@_column _ 1 _ min _ dim }
7140   \l_@@_left_margin_dim
7141 \dim_add:cn
7142   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7143   \l_@@_right_margin_dim
7144 }

```

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).

```

7145 \cs_new_protected:Npn \@@_create_nodes:
7146 {
7147   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7148   {
7149     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7150     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7151       \@@_pgf_rect_node:nnnnn
7152       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7153       { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7154       { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7155       { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7156       { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7157       \str_if_empty:NF \l_@@_name_str
7158       {
7159         \pgfnodealias
7160         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7161         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7162       }
7163     }
7164   }
7165   \int_step_inline:nn { \c@iRow }
7166   {
7167     \pgfnodealias
7168     { \@@_env: - ##1 - last \l_@@_suffix_tl }
7169     { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7170   }
7171   \int_step_inline:nn { \c@jCol }
7172   {
7173     \pgfnodealias
7174     { \@@_env: - last - ##1 \l_@@_suffix_tl }
7175     { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7176   }

```

```

7177 \pgfnodealias % added 2025-04-05
7178 { \@@_env: - last - last \l_@@_suffix_tl }
7179 { \@@_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 .

```

7180 \seq_map_pairwise_function:NNN
7181 \g_@@_multicolumn_cells_seq
7182 \g_@@_multicolumn_sizes_seq
7183 \@@_node_for_multicolumn:nn
7184 }

7185 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7186 {
7187   \cs_set_nopar:Npn \@@_i: { #1 }
7188   \cs_set_nopar:Npn \@@_j: { #2 }
7189 }

```

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”).

```

7190 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7191 {
7192   \@@_extract_coords_values: #1 \q_stop
7193   \@@_pgf_rect_node:nnnnn
7194   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7195   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7196   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7197   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: + #2 - 1 } _ max _ dim } }
7198   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7199   \str_if_empty:NF \l_@@_name_str
7200   {
7201     \pgfnodealias
7202     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7203     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7204   }
7205 }

```

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).

```

7206 \keys_define:nn { nicematrix / Block / FirstPass }
7207 {
7208   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7209           \bool_set_true:N \l_@@_p_block_bool ,
7210   j .value_forbidden:n = true ,
7211   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7212   l .value_forbidden:n = true ,
7213   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7214   r .value_forbidden:n = true ,
7215   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7216   c .value_forbidden:n = true ,

```

```

7217 L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7218 L .value_forbidden:n = true ,
7219 R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7220 R .value_forbidden:n = true ,
7221 C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7222 C .value_forbidden:n = true ,
7223 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7224 t .value_forbidden:n = true ,
7225 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7226 T .value_forbidden:n = true ,
7227 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7228 b .value_forbidden:n = true ,
7229 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7230 B .value_forbidden:n = true ,
7231 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7232 m .value_forbidden:n = true ,
7233 v-center .meta:n = m ,
7234 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7235 p .value_forbidden:n = true ,
7236 color .code:n =
7237   \@@_color:n { #1 }
7238   \tl_set_rescan:Nnn
7239     \l_@@_draw_tl
7240     { \char_set_catcode_other:N ! }
7241     { #1 } ,
7242 color .value_required:n = true ,
7243 respect-arraystretch .code:n =
7244   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7245 respect-arraystretch .value_forbidden:n = true ,
7246 }

```

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.

```

7247 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7248 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7249 {

```

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).

```

7250   \tl_if_blank:nTF { #2 }
7251   { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7252   {
7253     \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7254     \@@_Block_i_czech:w \@@_Block_i:w
7255     #2 \q_stop
7256   }
7257   { #1 } { #3 } { #4 }
7258   \ignorespaces
7259 }

```

With the following construction, we extract the values of *i* and *j* in the first mandatory argument of the command.

```

7260 \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`.

```

7261 {
7262   \char_set_catcode_active:N -
7263   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7264 }

```

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.

```

7265 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7266 {

```

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).

```

7267 \bool_lazy_or:nnTF
7268 { \tl_if_blank_p:n { #1 } }
7269 { \str_if_eq_p:ee { * } { #1 } }
7270 { \int_set:Nn \l_tmpa_int { 100 } }
7271 { \int_set:Nn \l_tmpa_int { #1 } }
7272 \bool_lazy_or:nnTF
7273 { \tl_if_blank_p:n { #2 } }
7274 { \str_if_eq_p:ee { * } { #2 } }
7275 { \int_set:Nn \l_tmpb_int { 100 } }
7276 { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7277 \int_compare:nNnTF { \l_tmpb_int } = { \c_one_int }
7278 {
7279   \tl_if_empty:NNTF \l_@@_hpos_cell_tl
7280   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7281   { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7282 }
7283 { \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.

```

7284 \keys_set_known:n { nicematrix / Block / FirstPass } { #3 }
7285 \tl_set:Nx \l_tmpa_tl
7286 {
7287   { \int_use:N \c@iRow }
7288   { \int_use:N \c@jCol }
7289   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7290   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7291 }

```

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).

```

7292 \bool_set_false:N \l_tmpa_bool
7293 \bool_if:NT \l_@@_amp_in_blocks_bool

```

\tl_if_in:nnT is slightly faster than \str_if_in:nnT.

```

7294 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7295 \bool_case:nF
7296 {
7297   \l_tmpa_bool { \@@_Block_vii:eennn }
7298   \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.

```

7299         \l_@@_X_bool                                { \@@_Block_v:eennn }
7300         { \tl_if_empty_p:n { #5 } }                    { \@@_Block_v:eennn }
7301         { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7302         { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7303     }
7304     { \@@_Block_v:eennn }
7305     { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7306 }

```

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.

```

7307 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7308 {
7309     \int_gincr:N \g_@@_block_box_int
7310     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7311     {
7312         \tl_gput_right:Ne \g_@@_pre_code_after_tl
7313         {
7314             \@@_actually_diagbox:nnnnnn
7315             { \int_use:N \c@iRow }
7316             { \int_use:N \c@jCol }
7317             { \int_eval:n { \c@iRow + #1 - 1 } }
7318             { \int_eval:n { \c@jCol + #2 - 1 } }
7319             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7320             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7321         }
7322     }
7323     \box_gclear_new:c
7324     { g_@@_block_box_int _ \int_use:N \g_@@_block_box_int _ box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7325     \hbox_gset:cn
7326     { g_@@_block_box_int _ \int_use:N \g_@@_block_box_int _ box }
7327     {

```

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`).

```

7328         \tl_if_empty:NTF \l_@@_color_tl
7329         { \int_compare:nNnT { #2 } = { \c_one_int } { \set@color } }
7330         { \@@_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`.

```

7331         \int_compare:nNnT { #1 } = { \c_one_int }

```

```

7332         {
7333             \int_if_zero:nTF { \c@iRow }
7334             {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7335         \cs_set_eq:NN \Block \@@_NullBlock:
7336         \l_@@_code_for_first_row_tl
7337     }
7338     {
7339         \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7340         {
7341             \cs_set_eq:NN \Block \@@_NullBlock:
7342             \l_@@_code_for_last_row_tl
7343         }
7344     }
7345     \g_@@_row_style_tl
7346 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7347 \@@_reset_arraystretch:
7348 \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7349 #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.).

```

7350 \@@_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}`.

```

7351 \bool_if:NTF \l_@@_tabular_bool
7352 {
7353     \bool_lazy_all:nTF
7354     {
7355         { \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.

```

7356      {
7357          ! \dim_compare_p:nNn
7358              { \l_@@_col_width_dim } < { \c_zero_dim }
7359      }
7360      { ! \g_@@_rotate_bool }
7361  }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7362      {
7363          \use:e
7364      {

```

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.

```

7365          \exp_not:N \begin { minipage }
7366          [ \str_lowercase:f \l_@@_vpos_block_str ]
7367          { \l_@@_col_width_dim }
7368          \str_case:on \l_@@_hpos_block_str
7369              { c \centering r \raggedleft l \raggedright }
7370      }
7371      #5
7372      \end { minipage }
7373  }

```

In the other cases, we use a `{tabular}`.

```

7374      {
7375          \bool_if:NT \c_@@_testphase_table_bool
7376          { \tagpdfsetup { table / tagging = presentation } }
7377          \use:e
7378          {
7379              \exp_not:N \begin { tabular }
7380              [ \str_lowercase:f \l_@@_vpos_block_str ]
7381              { @ { } \l_@@_hpos_block_str @ { } }
7382          }
7383          #5
7384          \end { tabular }
7385      }
7386  }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7387      {
7388          \c_math_toggle_token
7389          \use:e
7390          {
7391              \exp_not:N \begin { array }
7392              [ \str_lowercase:f \l_@@_vpos_block_str ]
7393              { @ { } \l_@@_hpos_block_str @ { } }
7394          }
7395          #5
7396          \end { array }
7397          \c_math_toggle_token
7398      }
7399  }

```

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).

```

7400      \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.

```

7401     \int_compare:nNnT { #2 } = { \c_one_int }
7402     {
7403         \dim_gset:Nn \g_@@_blocks_wd_dim
7404         {
7405             \dim_max:nn
7406             { \g_@@_blocks_wd_dim }
7407             {
7408                 \box_wd:c
7409                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7410             }
7411         }
7412     }

```

If we are in a mono-row block we take into account the height and the depth of that block for the height and the depth of the row, excepted when the block uses explicitly an option of vertical position T or B. Remind that if the user has not used a key for the vertical position of the block, then `\l_@@_vpos_block_str` remains empty.

```

7413     \int_compare:nNnT { #1 } = { \c_one_int }
7414     {
7415         \bool_lazy_any:nT
7416         {
7417             { \str_if_empty_p:N \l_@@_vpos_block_str }
7418             { \str_if_eq_p:ee \l_@@_vpos_block_str { t } }
7419             { \str_if_eq_p:ee \l_@@_vpos_block_str { b } }
7420         }
7421         { \@@_adjust_blocks_ht_dp: }
7422     }
7423     \seq_gput_right:Ne \g_@@_blocks_seq
7424     {
7425         \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.

```

7426     {
7427         \exp_not:n { #3 } ,
7428         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7429         \bool_if:NT \g_@@_rotate_bool
7430         {
7431             \bool_if:NTF \g_@@_rotate_c_bool
7432             { m }
7433             {
7434                 \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7435                 { T }
7436             }
7437         }
7438     }
7439     {
7440         \box_use_drop:c
7441         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7442     }
7443 }
7444 \bool_set_false:N \g_@@_rotate_c_bool
7445 }

7446 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7447 {
7448     \dim_gset:Nn \g_@@_blocks_ht_dim
7449     {

```



```

7450     \dim_max:nn
7451     { \g_@@_blocks_ht_dim }
7452     {
7453         \box_ht:c
7454         { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7455     }
7456 }
7457 \dim_gset:Nn \g_@@_blocks_dp_dim
7458 {
7459     \dim_max:nn
7460     { \g_@@_blocks_dp_dim }
7461     {
7462         \box_dp:c
7463         { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7464     }
7465 }
7466 }

7467 \cs_new:Npn \@@_adjust_hpos_rotate:
7468 {
7469     \bool_if:NT \g_@@_rotate_bool
7470     {
7471         \str_set:Nc \l_@@_hpos_block_str
7472         {
7473             \bool_if:NTF \g_@@_rotate_c_bool
7474             { c }
7475             {
7476                 \str_case:onF \l_@@_vpos_block_str
7477                 { b l B l t r T r }
7478                 {
7479                     \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
7480                     { r }
7481                     { l }
7482                 }
7483             }
7484         }
7485     }
7486 }
7487 \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*.

```

7488 \cs_new_protected:Npn \@@_rotate_box_of_block:
7489 {
7490     \box_grotate:cn
7491     { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7492     { 90 }
7493     \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7494     {
7495         \vbox_gset_top:cn
7496         { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7497         {
7498             \skip_vertical:n { 0.8 ex }
7499             \box_use:c
7500             { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7501         }
7502     }
7503     \bool_if:NT \g_@@_rotate_c_bool
7504     {
7505         \hbox_gset:cn
7506         { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7507         {

```

```

7508         \c_math_toggle_token
7509         \vcenter
7510         {
7511             \box_use:c
7512             { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7513         }
7514         \c_math_toggle_token
7515     }
7516 }
7517 }

```

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.

```

7518 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7519 {
7520     \seq_gput_right:Ne \g_@@_blocks_seq
7521     {
7522         \l_tmpa_tl
7523         { \exp_not:n { #3 } }
7524         {
7525             \bool_if:NTF \l_@@_tabular_bool
7526             {
7527                 \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7528         \@@_reset_arraystretch:
7529         \exp_not:n
7530         {
7531             \dim_zero:N \extrarowheight
7532             #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).

```

7533         \bool_if:NT \c_@@_testphase_table_bool
7534         { \tag_stop:n { table } }
7535         \use:e
7536         {
7537             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7538             { @ { } \l_@@_hpos_block_str @ { } }
7539         }
7540         #5
7541         \end { tabular }
7542     }
7543     \group_end:
7544 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7545     {
7546         \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7547         \@@_reset_arraystretch:
7548         \exp_not:n
7549         {
7550             \dim_zero:N \extrarowheight
7551             #4

```

```

7552         \c_math_toggle_token
7553         \use:e
7554         {
7555             \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7556             { @ { } \l_@@_hpos_block_str @ { } }
7557         }
7558         #5
7559         \end { array }
7560         \c_math_toggle_token
7561     }
7562     \group_end:
7563 }
7564 }
7565 }
7566 }
7567 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7568 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7569 {
7570     \seq_gput_right:Ne \g_@@_blocks_seq
7571     {
7572         \l_tmpa_tl
7573         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7574         { { \exp_not:n { #4 #5 } } }
7575     }
7576 }
7577 \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`.

```

7578 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7579 {
7580     \seq_gput_right:Ne \g_@@_blocks_seq
7581     {
7582         \l_tmpa_tl
7583         { \exp_not:n { #3 } }
7584         { \exp_not:n { #4 #5 } }
7585     }
7586 }
7587 \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).

```

7588 \keys_define:nn { nicematrix / Block / SecondPass }
7589 {
7590     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7591     ampersand-in-blocks .default:n = true ,
7592     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7593     tikz .code:n =
7594         \IfPackageLoadedTF { tikz }
7595         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7596         { \@@_error:n { tikz-key-without-tikz } } ,
7597     tikz .value_required:n = true ,
7598     fill .code:n =
7599         \tl_set_rescan:Nnn
7600         \l_@@_fill_tl

```

```

7601     { \char_set_catcode_other:N ! }
7602     { #1 } ,
7603     fill .value_required:n = true ,
7604     opacity .tl_set:N = \l_@@_opacity_tl ,
7605     opacity .value_required:n = true ,
7606     draw .code:n =
7607         \tl_set_rescan:Nnn
7608         \l_@@_draw_tl
7609         { \char_set_catcode_other:N ! }
7610         { #1 } ,
7611     draw .default:n = default ,
7612     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7613     rounded-corners .default:n = 4 pt ,
7614     color .code:n =
7615         \@@_color:n { #1 }
7616         \tl_set_rescan:Nnn
7617         \l_@@_draw_tl
7618         { \char_set_catcode_other:N ! }
7619         { #1 } ,
7620     borders .clist_set:N = \l_@@_borders_clist ,
7621     borders .value_required:n = true ,
7622     hvlines .meta:n = { vlines , hlines } ,
7623     vlines .bool_set:N = \l_@@_vlines_block_bool ,
7624     vlines .default:n = true ,
7625     hlines .bool_set:N = \l_@@_hlines_block_bool ,
7626     hlines .default:n = true ,
7627     line-width .dim_set:N = \l_@@_line_width_dim ,
7628     line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7629     j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7630         \bool_set_true:N \l_@@_p_block_bool ,
7631     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7632     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7633     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7634     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7635         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7636     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7637         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7638     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7639         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7640     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7641     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7642     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7643     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7644     m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7645     m .value_forbidden:n = true ,
7646     v-center .meta:n = m ,
7647     p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7648     p .value_forbidden:n = true ,
7649     name .tl_set:N = \l_@@_block_name_str ,
7650     name .value_required:n = true ,
7651     name .initial:n = ,
7652     respect-arraystretch .code:n =
7653         \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7654     respect-arraystretch .value_forbidden:n = true ,
7655     transparent .bool_set:N = \l_@@_transparent_bool ,
7656     transparent .default:n = true ,
7657     transparent .initial:n = false ,
7658     unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7659 }

```

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.

```

7660 \cs_new_protected:Npn \@@_draw_blocks:
7661 {
7662   \bool_if:nTF { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }
7663     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7664     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7665   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7666 }
7667 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7668 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7669   \int_zero:N \l_@@_last_row_int
7670   \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”).

```

7671   \int_compare:nNnTF { #3 } > { 98 }
7672     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7673     { \int_set:Nn \l_@@_last_row_int { #3 } }
7674   \int_compare:nNnTF { #4 } > { 98 }
7675     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7676     { \int_set:Nn \l_@@_last_col_int { #4 } }
7677   \int_compare:nNnTF { \l_@@_last_col_int } > { \g_@@_col_total_int }
7678     {
7679       \bool_lazy_and:nnTF
7680         { \l_@@_preamble_bool }
7681         {
7682           \int_compare_p:n
7683             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7684         }
7685       {
7686         \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7687         \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7688         \@@_msg_redirect_name:nn { columns-not-used } { none }
7689       }
7690       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7691     }
7692   {
7693     \int_compare:nNnTF { \l_@@_last_row_int } > { \g_@@_row_total_int }
7694       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7695       {
7696         \@@_Block_v:nneenn
7697           { #1 }
7698           { #2 }
7699           { \int_use:N \l_@@_last_row_int }
7700           { \int_use:N \l_@@_last_col_int }
7701           { #5 }
7702           { #6 }
7703       }
7704     }
7705   }

```

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

```

7706 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7707 {

```

The group is for the keys.

```

7708 \group_begin:
7709 \int_compare:nNt { #1 } = { #3 }
7710 { \str_set:Nn \l_@@_vpos_block_str { t } }
7711 \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`.

```

7712 \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7713 \bool_lazy_and:nnT
7714 { \l_@@_vlines_block_bool }
7715 { ! \l_@@_ampersand_bool }
7716 {
7717   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7718   {
7719     \@@_vlines_block:nnn
7720     { \exp_not:n { #5 } }
7721     { #1 - #2 }
7722     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7723   }
7724 }
7725 \bool_if:NT \l_@@_hlines_block_bool
7726 {
7727   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7728   {
7729     \@@_hlines_block:nnn
7730     { \exp_not:n { #5 } }
7731     { #1 - #2 }
7732     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7733   }
7734 }
7735 \bool_if:NF \l_@@_transparent_bool
7736 {
7737   \bool_lazy_and:nnF { \l_@@_vlines_block_bool } { \l_@@_hlines_block_bool }
7738   {

```

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).

```

7739 \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7740 { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7741 }
7742 }

```

```

7743 \tl_if_empty:NF \l_@@_draw_tl
7744 {
7745   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7746   { \@@_error:n { hlines-with-color } }
7747   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7748   {
7749     \@@_stroke_block:nnn

```

#5 are the options

```

7750 { \exp_not:n { #5 } }
7751 { #1 - #2 }
7752 { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7753 }
7754 \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7755 { { #1 } { #2 } { #3 } { #4 } }
7756 }
7757 \clist_if_empty:NF \l_@@_borders_clist
7758 {
7759   \tl_gput_right:Ne \g_nicematrix_code_after_tl

```

```

7760         {
7761             \@@_stroke_borders_block:nnn
7762             { \exp_not:n { #5 } }
7763             { #1 - #2 }
7764             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7765         }
7766     }
7767 \tl_if_empty:NF \l_@@_fill_tl
7768 {
7769     \@@_add_opacity_to_fill:
7770     \tl_gput_right:Ne \g_@@_pre_code_before_tl
7771     {
7772         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7773         { #1 - #2 }
7774         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7775         { \dim_use:N \l_@@_rounded_corners_dim }
7776     }
7777 }
7778 \seq_if_empty:NF \l_@@_tikz_seq
7779 {
7780     \tl_gput_right:Ne \g_nicematrix_code_before_tl
7781     {
7782         \@@_block_tikz:nnnnn
7783         { \seq_use:Nn \l_@@_tikz_seq { , } }
7784         { #1 }
7785         { #2 }
7786         { \int_use:N \l_@@_last_row_int }
7787         { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7788     }
7789 }
7790 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7791 {
7792     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7793     {
7794         \@@_actually_diagbox:nnnnn
7795         { #1 }
7796         { #2 }
7797         { \int_use:N \l_@@_last_row_int }
7798         { \int_use:N \l_@@_last_col_int }
7799         { \exp_not:n { ##1 } }
7800         { \exp_not:n { ##2 } }
7801     }
7802 }

```

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.

```

7803 \pgfpicture
7804 \pgfrememberpicturepositiononpagetrue
7805 \pgf@relevantforpicturesizefalse
7806 \@@_qpoint:n { row - #1 }
7807 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7808 \@@_qpoint:n { col - #2 }
7809 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7810 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7811 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7812 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7813 \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.

```

7814 \@@_pgf_rect_node:nnnnn
7815 { \@@_env: - #1 - #2 - block }
7816 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7817 \str_if_empty:NF \l_@@_block_name_str
7818 {
7819 \pgfnodealias
7820 { \@@_env: - \l_@@_block_name_str }
7821 { \@@_env: - #1 - #2 - block }
7822 \str_if_empty:NF \l_@@_name_str
7823 {
7824 \pgfnodealias
7825 { \l_@@_name_str - \l_@@_block_name_str }
7826 { \@@_env: - #1 - #2 - block }
7827 }
7828 }

```

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.

```

7829 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7830 {
7831 \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.

```

7832 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7833 {

```

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.

```

7834 \cs_if_exist:cT
7835 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7836 {
7837 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7838 {
7839 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7840 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7841 }
7842 }
7843 }

```


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.

```

7844     \dim_compare:nNtT { \l_tmpb_dim } = { \c_max_dim }
7845     {
7846         \@@_qpoint:n { col - #2 }
7847         \dim_set_eq:NN \l_tmpb_dim \pgf@x
7848     }
7849     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7850     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7851     {
7852         \cs_if_exist:cT
7853         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7854         {
7855             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7856             {
7857                 \pgfpointanchor
7858                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7859                 { east }
7860                 \dim_set:Nn \l_@@_tmpd_dim
7861                 { \dim_max:nn { \l_@@_tmpd_dim } { \pgf@x } }
7862             }
7863         }
7864     }
7865     \dim_compare:nNtT { \l_@@_tmpd_dim } = { - \c_max_dim }
7866     {
7867         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7868         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7869     }
7870     \@@_pgf_rect_node:nnnnn
7871     { \@@_env: - #1 - #2 - block - short }
7872     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7873 }

```

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.

```

7874     \bool_if:NT \l_@@_medium_nodes_bool
7875     {
7876         \@@_pgf_rect_node:nnn
7877         { \@@_env: - #1 - #2 - block - medium }
7878         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7879         {
7880             \pgfpointanchor
7881             { \@@_env:
7882               - \int_use:N \l_@@_last_row_int
7883               - \int_use:N \l_@@_last_col_int - medium
7884             }
7885             { south-east }
7886         }
7887     }
7888     \endpgfpicture
7889
7890     \bool_if:NtF \l_@@_ampersand_bool
7891     {
7892         \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7893         \int_zero_new:N \l_@@_split_int
7894         \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7895         \pgfpicture
7896         \pgfrememberpicturepositiononpagetrue
7897         \pgf@relevantforpicturesizefalse
7898
7899         \@@_qpoint:n { row - #1 }

```

```

7900 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7901 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7902 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7903 \@@_qpoint:n { col - #2 }
7904 \dim_set_eq:NN \l_tmpa_dim \pgf@x
7905 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7906 \dim_set:Nn \l_tmpb_dim
7907 { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7908 \bool_lazy_or:nnT
7909 { \l_@@_vlines_block_bool }
7910 { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
7911 {
7912   \int_step_inline:nn { \l_@@_split_int - 1 }
7913   {
7914     \pgfpathmoveto
7915     {
7916       \pgfpoint
7917       { \l_tmpa_dim + ##1 \l_tmpb_dim }
7918       \l_@@_tmpc_dim
7919     }
7920     \pgfpathlineto
7921     {
7922       \pgfpoint
7923       { \l_tmpa_dim + ##1 \l_tmpb_dim }
7924       \l_@@_tmpd_dim
7925     }
7926     \CT@arc@
7927     \pgfsetlinewidth { 1.1 \arrayrulewidth }
7928     \pgfsetrectcap
7929     \pgfusepathqstroke
7930   }
7931 }
7932 \@@_qpoint:n { row - #1 - base }
7933 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7934 \int_step_inline:nn { \l_@@_split_int }
7935 {
7936   \group_begin:
7937   \dim_set:Nn \col@sep
7938   { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
7939   \pgftransformshift
7940   {
7941     \pgfpoint
7942     {
7943       \l_tmpa_dim + ##1 \l_tmpb_dim -
7944       \str_case:on \l_@@_hpos_block_str
7945       {
7946         l { \l_tmpb_dim + \col@sep }
7947         c { 0.5 \l_tmpb_dim }
7948         r { \col@sep }
7949       }
7950     }
7951     { \l_@@_tmpc_dim }
7952   }
7953   \pgfset { inner~sep = \c_zero_dim }
7954   \pgfnode
7955   { rectangle }
7956   {
7957     \str_case:on \l_@@_hpos_block_str
7958     {
7959       c { base }
7960       l { base~west }
7961       r { base~east }
7962     }
7963   }

```

```

7963     }
7964     { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
7965     \group_end:
7966   }
7967   \endpgfpicture
7968 }

```

Now the case where there is no ampersand & in the content of the block.

```

7969 {
7970   \bool_if:NTF \l_@@_p_block_bool
7971   {

```

When the final user has used the key p, we have to compute the width.

```

7972     \pgfpicture
7973     \pgfrememberpicturepositiononpagetrue
7974     \pgf@relevantforpicturesizefalse
7975     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7976     {
7977       \@@_qpoint:n { col - #2 }
7978       \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7979       \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7980     }
7981     {
7982       \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7983       \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7984       \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7985     }
7986     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7987   \endpgfpicture
7988   \hbox_set:Nn \l_@@_cell_box
7989   {
7990     \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
7991     { \g_tmpb_dim }
7992     \str_case:on \l_@@_hpos_block_str
7993     { c \centering r \raggedleft l \raggedright j { } }
7994     #6
7995     \end { minipage }
7996   }
7997 }
7998 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7999 \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.

```

8000   \pgfpicture
8001   \pgfrememberpicturepositiononpagetrue
8002   \pgf@relevantforpicturesizefalse
8003   \bool_lazy_any:nTF
8004   {
8005     { \str_if_empty_p:N \l_@@_vpos_block_str }
8006     { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
8007     { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
8008     { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
8009   }
8010   {

```

If we are in the first column, we must put the block as if it was with the key r.

```

8011     \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.

```

8012     \bool_if:nT \g_@@_last_col_found_bool
8013     {
8014       \int_compare:nNnT { #2 } = { \g_@@_col_total_int }

```

```

8015         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8016     }

```

\l_tmpa_tl will contain the anchor of the PGF node which will be used.

```

8017     \tl_set:Ne \l_tmpa_tl
8018     {
8019         \str_case:on \l_@@_vpos_block_str
8020     {

```

We recall that \l_@@_vpos_block_str is empty when the user has not used a key for the vertical position of the block.

```

8021         { } {
8022             \str_case:on \l_@@_hpos_block_str
8023             {
8024                 c { center }
8025                 l { west }
8026                 r { east }
8027                 j { center }
8028             }
8029         }
8030     c {
8031         \str_case:on \l_@@_hpos_block_str
8032         {
8033             c { center }
8034             l { west }
8035             r { east }
8036             j { center }
8037         }
8038     }
8039     T {
8040         \str_case:on \l_@@_hpos_block_str
8041         {
8042             c { north }
8043             l { north~west }
8044             r { north~east }
8045             j { north }
8046         }
8047     }
8048 }
8049 }
8050 B {
8051     \str_case:on \l_@@_hpos_block_str
8052     {
8053         c { south }
8054         l { south~west }
8055         r { south~east }
8056         j { south }
8057     }
8058 }
8059 }
8060 }
8061 }
8062 \pgftransformshift
8063 {
8064     \pgfpointanchor
8065     {
8066         \@@_env: - #1 - #2 - block
8067         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8068     }
8069     { \l_tmpa_tl }
8070 }
8071 \pgfset { inner~sep = \c_zero_dim }
8072 \pgfnode
8073 { rectangle }

```

```

8074         { \l_tmpa_tl }
8075         { \box_use_drop:N \l_@@_cell_box } { } { }
8076     }

```

End of the case when `\l_@@_vpos_block_str` is equal to c, T or B. Now, the other cases.

```

8077     {
8078         \pgfextracty \l_tmpa_dim
8079         {
8080             \@@_qpoint:n
8081             {
8082                 row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
8083                 - base
8084             }
8085         }
8086         \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

8087         \pgfpointanchor
8088         {
8089             \@@_env: - #1 - #2 - block
8090             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8091         }
8092         {
8093             \str_case:on \l_@@_hpos_block_str
8094             {
8095                 c { center }
8096                 l { west }
8097                 r { east }
8098                 j { center }
8099             }
8100         }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8101         \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8102         \pgfset { inner~sep = \c_zero_dim }
8103         \pgfnode
8104         { rectangle }
8105         {
8106             \str_case:on \l_@@_hpos_block_str
8107             {
8108                 c { base }
8109                 l { base~west }
8110                 r { base~east }
8111                 j { base }
8112             }
8113         }
8114         { \box_use_drop:N \l_@@_cell_box } { } { }
8115     }
8116     \endpgfpicture
8117 }
8118 \group_end:
8119 }
8120 \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).

```

8121 \cs_set_protected:Npn \@@_fill:nnnnn #1 #2 #3 #4 #5
8122 {
8123     \pgfpicture
8124     \pgfrememberpicturepositiononpagetrue
8125     \pgf@relevantforpicturesizefalse
8126     \pgfpathrectanglecorners
8127     { \pgfpoint { #2 } { #3 } }

```

```

8128     { \pgfpoint { #4 } { #5 } }
8129     \pgfsetfillcolor { #1 }
8130     \pgfusepath { fill }
8131     \endpgfpicture
8132 }

```

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).

```

8133 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8134 {
8135     \tl_if_empty:NF \l_@@_opacity_tl
8136     {
8137         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8138             {
8139                 \tl_set:Nc \l_@@_fill_tl
8140                 {
8141                     [ opacity = \l_@@_opacity_tl ,
8142                     \tl_tail:o \l_@@_fill_tl
8143                 }
8144             }
8145             {
8146                 \tl_set:Nc \l_@@_fill_tl
8147                 { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8148             }
8149         }
8150     }

```

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).

```

8151 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8152 {
8153     \group_begin:
8154     \tl_clear:N \l_@@_draw_tl
8155     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8156     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8157     \pgfpicture
8158     \pgfrememberpicturepositiononpagetrue
8159     \pgf@relevantforpicturesizefalse
8160     \tl_if_empty:NF \l_@@_draw_tl
8161     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8162         \tl_if_eq:NnTF \l_@@_draw_tl { default }
8163         { \CT@arc@ }
8164         { \@@_color:o \l_@@_draw_tl }
8165     }
8166     \pgfsetcornersarced
8167     {
8168         \pgfpoint
8169         { \l_@@_rounded_corners_dim }
8170         { \l_@@_rounded_corners_dim }
8171     }
8172     \@@_cut_on_hyphen:w #2 \q_stop
8173     \int_compare:nNnF { \l_tmpa_tl } > { \c@iRow }
8174     {
8175         \int_compare:nNnF { \l_tmpb_tl } > { \c@jCol }
8176         {
8177             \@@_qpoint:n { row - \l_tmpa_tl }
8178             \dim_set_eq:NN \l_tmpb_dim \pgf@y

```

```

8179 \@@_qpoint:n { col - \l_tmpb_tl }
8180 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8181 \@@_cut_on_hyphen:w #3 \q_stop
8182 \int_compare:nNnT { \l_tmpa_tl } > { \c@iRow }
8183 { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8184 \int_compare:nNnT { \l_tmpb_tl } > { \c@jCol }
8185 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8186 \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8187 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8188 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8189 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8190 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8191 \pgfpathrectanglecorners
8192 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8193 { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8194 \dim_compare:nNnTF { \l_@@_rounded_corners_dim } = { \c_zero_dim }
8195 { \pgfusepathqstroke }
8196 { \pgfusepath { stroke } }
8197 }
8198 }
8199 \endpgfpicture
8200 \group_end:
8201 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8202 \keys_define:nn { nicematrix / BlockStroke }
8203 {
8204   color .tl_set:N = \l_@@_draw_tl ,
8205   draw .code:n =
8206     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8207   draw .default:n = default ,
8208   line-width .dim_set:N = \l_@@_line_width_dim ,
8209   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8210   rounded-corners .default:n = 4 pt
8211 }

```

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).

```

8212 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8213 {
8214   \group_begin:
8215   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8216   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8217   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8218   \@@_cut_on_hyphen:w #2 \q_stop
8219   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8220   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8221   \@@_cut_on_hyphen:w #3 \q_stop
8222   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8223   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8224   \int_step_inline:nnn { \l_@@_tmpd_tl } { \l_tmpb_tl }
8225   {
8226     \use:e
8227     {
8228       \@@_vline:n
8229       {
8230         position = ##1 ,
8231         start = \l_@@_tmpc_tl ,
8232         end = \int_eval:n { \l_tmpa_tl - 1 } ,
8233         total-width = \dim_use:N \l_@@_line_width_dim
8234       }
8235     }

```

```

8236     }
8237   \group_end:
8238 }

8239 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8240 {
8241   \group_begin:
8242   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8243   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8244   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8245   \@@_cut_on_hyphen:w #2 \q_stop
8246   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8247   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8248   \@@_cut_on_hyphen:w #3 \q_stop
8249   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8250   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8251   \int_step_inline:nnn { \l_@@_tmpc_tl } { \l_tmpa_tl }
8252   {
8253     \use:e
8254     {
8255       \@@_hline:n
8256       {
8257         position = ##1 ,
8258         start = \l_@@_tmpd_tl ,
8259         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8260         total-width = \dim_use:N \l_@@_line_width_dim
8261       }
8262     }
8263   }
8264   \group_end:
8265 }

```

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).

```

8266 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8267 {
8268   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8269   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8270   \dim_compare:nNnTF { \l_@@_rounded_corners_dim } > { \c_zero_dim }
8271   { \@@_error:n { borders~forbidden } }
8272   {
8273     \tl_clear_new:N \l_@@_borders_tikz_tl
8274     \keys_set:no
8275     { nicematrix / OnlyForTikzInBorders }
8276     \l_@@_borders_clist
8277     \@@_cut_on_hyphen:w #2 \q_stop
8278     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8279     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8280     \@@_cut_on_hyphen:w #3 \q_stop
8281     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8282     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8283     \@@_stroke_borders_block_i:
8284   }
8285 }

8286 \hook_gput_code:nnn { begindocument } { . }
8287 {
8288   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8289   {
8290     \c_@@_pgfortikzpicture_tl
8291     \@@_stroke_borders_block_ii:
8292     \c_@@_endpgfortikzpicture_tl
8293   }

```



```

8294 }
8295 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8296 {
8297   \pgfrememberpicturepositiononpagetrue
8298   \pgf@relevantforpicturesizefalse
8299   \CT@arc@
8300   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8301   \clist_if_in:NnT \l_@@_borders_clist { right }
8302     { \@@_stroke_vertical:n \l_tmpb_tl }
8303   \clist_if_in:NnT \l_@@_borders_clist { left }
8304     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8305   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8306     { \@@_stroke_horizontal:n \l_tmpa_tl }
8307   \clist_if_in:NnT \l_@@_borders_clist { top }
8308     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8309 }
8310 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8311 {
8312   tikz .code:n =
8313     \cs_if_exist:NTF \tikzpicture
8314       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8315       { \@@_error:n { tikz-in-borders-without-tikz } } ,
8316   tikz .value_required:n = true ,
8317   top .code:n = ,
8318   bottom .code:n = ,
8319   left .code:n = ,
8320   right .code:n = ,
8321   unknown .code:n = \@@_error:n { bad~border }
8322 }

```

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).

```

8323 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8324 {
8325   \@@_qpoint:n \l_@@_tmpc_tl
8326   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8327   \@@_qpoint:n \l_tmpa_tl
8328   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8329   \@@_qpoint:n { #1 }
8330   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8331   {
8332     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8333     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8334     \pgfusepathqstroke
8335   }
8336   {
8337     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8338     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8339   }
8340 }

```

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).

```

8341 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8342 {
8343   \@@_qpoint:n \l_@@_tmpd_tl
8344   \clist_if_in:NnTF \l_@@_borders_clist { left }
8345     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8346     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8347   \@@_qpoint:n \l_tmpb_tl
8348   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8349   \@@_qpoint:n { #1 }

```

```

8350 \tl_if_empty:NTF \l_@@_borders_tikz_tl
8351 {
8352   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8353   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8354   \pgfusepathqstroke
8355 }
8356 {
8357   \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8358   ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8359 }
8360 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

8361 \keys_define:nn { nicematrix / BlockBorders }
8362 {
8363   borders .clist_set:N = \l_@@_borders_clist ,
8364   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8365   rounded-corners .default:n = 4 pt ,
8366   line-width .dim_set:N = \l_@@_line_width_dim
8367 }

```

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.

```

8368 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8369 {
8370   \begin { tikzpicture }
8371   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because **#5** is a list of lists.

```

8372   \clist_map_inline:nn { #1 }
8373   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8374   \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8375   \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8376   (
8377     [
8378       xshift = \dim_use:N \l_@@_offset_dim ,
8379       yshift = - \dim_use:N \l_@@_offset_dim
8380     ]
8381     #2 -| #3
8382   )
8383   rectangle
8384   (
8385     [
8386       xshift = - \dim_use:N \l_@@_offset_dim ,
8387       yshift = \dim_use:N \l_@@_offset_dim
8388     ]
8389     \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8390   ) ;
8391 }
8392 \end { tikzpicture }
8393 }
8394 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

```

```

8395 \keys_define:nn { nicematrix / SpecialOffset }
8396 { 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.

```
8397 \cs_new_protected:Npn \@@_NullBlock:
8398 { \@@_collect_options:n { \@@_NullBlock_i: } }
8399 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8400 { }
```

27 How to draw the dotted lines transparently

```
8401 \cs_set_protected:Npn \@@_renew_matrix:
8402 {
8403   \RenewDocumentEnvironment { pmatrix } { } {
8404     { \pNiceMatrix }
8405     { \endpNiceMatrix }
8406   \RenewDocumentEnvironment { vmatrix } { } {
8407     { \vNiceMatrix }
8408     { \endvNiceMatrix }
8409   \RenewDocumentEnvironment { Vmatrix } { } {
8410     { \VNiceMatrix }
8411     { \endVNiceMatrix }
8412   \RenewDocumentEnvironment { bmatrix } { } {
8413     { \bNiceMatrix }
8414     { \endbNiceMatrix }
8415   \RenewDocumentEnvironment { Bmatrix } { } {
8416     { \BNiceMatrix }
8417     { \endBNiceMatrix }
8418 }
```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```
8419 \keys_define:nn { nicematrix / Auto }
8420 {
8421   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8422   columns-type .value_required:n = true ,
8423   l .meta:n = { columns-type = l } ,
8424   r .meta:n = { columns-type = r } ,
8425   c .meta:n = { columns-type = c } ,
8426   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8427   delimiters / color .value_required:n = true ,
8428   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8429   delimiters / max-width .default:n = true ,
8430   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8431   delimiters .value_required:n = true ,
8432   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8433   rounded-corners .default:n = 4 pt
8434 }
8435 \NewDocumentCommand \AutoNiceMatrixWithDelims
8436 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8437 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
8438 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8439 {
```

The group is for the protection of the keys.

```
8440   \group_begin:
8441   \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
```

```

8442 \use:e
8443 {
8444   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8445   { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8446   [ \exp_not:o \l_tmpa_tl ]
8447 }
8448 \int_if_zero:nT { \l_@@_first_row_int }
8449 {
8450   \int_if_zero:nT { \l_@@_first_col_int } { & }
8451   \prg_replicate:nn { #4 - 1 } { & }
8452   \int_compare:nNtT { \l_@@_last_col_int } > { -1 } { & } \\
8453 }
8454 \prg_replicate:nn { #3 }
8455 {
8456   \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).

```

8457   \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8458   \int_compare:nNtT { \l_@@_last_col_int } > { -1 } { & } \\
8459 }
8460 \int_compare:nNtT { \l_@@_last_row_int } > { -2 }
8461 {
8462   \int_if_zero:nT { \l_@@_first_col_int } { & }
8463   \prg_replicate:nn { #4 - 1 } { & }
8464   \int_compare:nNtT { \l_@@_last_col_int } > { -1 } { & } \\
8465 }
8466 \end { NiceArrayWithDelims }
8467 \group_end:
8468 }
8469 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
8470 {
8471   \cs_set_protected:cpn { #1 AutoNiceMatrix }
8472   {
8473     \bool_gset_true:N \g_@@_delims_bool
8474     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8475     \AutoNiceMatrixWithDelims { #2 } { #3 }
8476   }
8477 }

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8478 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8479 {
8480   \group_begin:
8481   \bool_gset_false:N \g_@@_delims_bool
8482   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8483   \group_end:
8484 }

```

29 The redefinition of the command \dotfill

```

8485 \cs_set_eq:NN \@@_old_dotfill: \dotfill
8486 \cs_new_protected:Npn \@@_dotfill:
8487 {

```

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}).

```

8488   \@@_old_dotfill:

```

```

8489 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8490 }

```

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 its 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`.

```

8491 \cs_new_protected:Npn \@@_dotfill_i:
8492 {
8493   \dim_compare:nNtT { \box_wd:N \l_@@_cell_box } = { \c_zero_dim }
8494     { \@@_old_dotfill: }
8495 }

```

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.

```

8496 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8497 {
8498   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8499   {
8500     \@@_actually_diagbox:nnnnnn
8501     { \int_use:N \c@iRow }
8502     { \int_use:N \c@jCol }
8503     { \int_use:N \c@iRow }
8504     { \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.

```

8505     { \g_@@_row_style_tl \exp_not:n { #1 } }
8506     { \g_@@_row_style_tl \exp_not:n { #2 } }
8507   }

```

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.

```

8508   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8509   {
8510     { \int_use:N \c@iRow }
8511     { \int_use:N \c@jCol }
8512     { \int_use:N \c@iRow }
8513     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8514     { }
8515   }
8516 }

```

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.

```

8517 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8518 {
8519   \pgfpicture
8520   \pgf@relevantforpicturesizefalse
8521   \pgfrememberpicturepositiononpagetrue
8522   \@@_qpoint:n { row - #1 }
8523   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8524   \@@_qpoint:n { col - #2 }

```

```

8525 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8526 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8527 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8528 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8529 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8530 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8531 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8532 {

```

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.

```

8533 \CT@arc@
8534 \pgfsetroundcap
8535 \pgfusepathqstroke
8536 }
8537 \pgfset { inner~sep = 1 pt }
8538 \pgfscope
8539 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8540 \pgfnode { rectangle } { south~west }
8541 {
8542 \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument `#5` is empty.

```

8543 \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8544 \end { minipage }
8545 }
8546 { }
8547 { }
8548 \endpgfscope
8549 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8550 \pgfnode { rectangle } { north~east }
8551 {
8552 \begin { minipage } { 20 cm }
8553 \raggedleft
8554 \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8555 \end { minipage }
8556 }
8557 { }
8558 { }
8559 \endpgfpicture
8560 }

```

31 The keyword `\CodeAfter`

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment `{@@-light-syntax}` on p. 86.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

8561 \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 `\.`

```

8562 \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`.

```

8563 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8564 {
8565   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8566   \@@_CodeAfter_iv:n
8567 }

```

We catch the argument of the command `\end` (in `#1`).

```

8568 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8569 {

```

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.

```

8570   \str_if_eq:eeTF \currenvir { #1 }
8571   { \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`.

```

8572   {
8573     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8574     \@@_CodeAfter_ii:n
8575   }
8576 }

```

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.

```

8577 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8578 {
8579   \pgfpicture
8580   \pgfrememberpicturepositiononpagetrue
8581   \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.

```

8582   \@@_qpoint:n { row - 1 }
8583   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8584   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8585   \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).

```

8586   \bool_if:nTF { #3 }
8587   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8588   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8589   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8590   {
8591     \cs_if_exist:cT
8592     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }

```

```

8593     {
8594         \pgfpointanchor
8595         { \@@_env: - ##1 - #2 }
8596         { \bool_if:nTF { #3 } { west } { east } }
8597         \dim_set:Nn \l_tmpa_dim
8598         {
8599             \bool_if:nTF { #3 }
8600             { \dim_min:nn }
8601             { \dim_max:nn }
8602             \l_tmpa_dim
8603             { \pgf@x }
8604         }
8605     }
8606 }

```

Now we can put the delimiter with a node of PGF.

```

8607     \pgfset { inner~sep = \c_zero_dim }
8608     \dim_zero:N \nulldelimiterspace
8609     \pgftransformshift
8610     {
8611         \pgfpoint
8612         { \l_tmpa_dim }
8613         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8614     }
8615     \pgfnode
8616     { rectangle }
8617     { \bool_if:nTF { #3 } { east } { west } }
8618     {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8619         \nullfont
8620         \c_math_toggle_token
8621         \@@_color:o \l_@@_delimiters_color_tl
8622         \bool_if:nTF { #3 } { \left #1 } { \left . }
8623         \vcenter
8624         {
8625             \nullfont
8626             \hrule \@height
8627                 \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8628                 \@depth \c_zero_dim
8629                 \@width \c_zero_dim
8630         }
8631         \bool_if:nTF { #3 } { \right . } { \right #1 }
8632         \c_math_toggle_token
8633     }
8634     { }
8635     { }
8636 \endpgfpicture
8637 }

```

33 The command `\SubMatrix`

```

8638 \keys_define:nn { nicematrix / sub-matrix }
8639 {
8640     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8641     extra-height .value_required:n = true ,
8642     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8643     left-xshift .value_required:n = true ,
8644     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8645     right-xshift .value_required:n = true ,
8646     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,

```



```

8647     xshift .value_required:n = true ,
8648     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8649     delimiters / color .value_required:n = true ,
8650     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8651     slim .default:n = true ,
8652     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8653     hlines .default:n = all ,
8654     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8655     vlines .default:n = all ,
8656     hvlines .meta:n = { hlines, vlines } ,
8657     hvlines .value_forbidden:n = true
8658   }
8659 \keys_define:nn { nicematrix }
8660 {
8661   SubMatrix .inherit:n = nicematrix / sub-matrix ,
8662   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8663   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8664   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8665 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8666 \keys_define:nn { nicematrix / SubMatrix }
8667 {
8668   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8669   delimiters / color .value_required:n = true ,
8670   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8671   hlines .default:n = all ,
8672   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8673   vlines .default:n = all ,
8674   hvlines .meta:n = { hlines, vlines } ,
8675   hvlines .value_forbidden:n = true ,
8676   name .code:n =
8677     \tl_if_empty:nTF { #1 }
8678     { \@@_error:n { Invalid-name } }
8679     {
8680       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8681       {
8682         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8683         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8684         {
8685           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8686           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8687         }
8688       }
8689       { \@@_error:n { Invalid-name } }
8690     } ,
8691   name .value_required:n = true ,
8692   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8693   rules .value_required:n = true ,
8694   code .tl_set:N = \l_@@_code_tl ,
8695   code .value_required:n = true ,
8696   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
8697 }

8698 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
8699 {
8700   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8701   {
8702     \SubMatrix { #1 } { #2 } { #3 } { #4 }
8703     [
8704       delimiters / color = \l_@@_delimiters_color_tl ,
8705       hlines = \l_@@_submatrix_hlines_clist ,

```

```

8706         vlines = \l_@@_submatrix_vlines_clist ,
8707         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8708         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8709         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8710         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8711         #5
8712     ]
8713 }
8714 \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8715 \ignorespaces
8716 }
8717 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8718 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8719 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8720 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8721 {
8722     \seq_gput_right:Ne \g_@@_submatrix_seq
8723     {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8724     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8725     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8726     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8727     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8728 }
8729 }

```

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).

```

8730 \NewDocumentCommand \@@_compute_i_j:nn
8731 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8732 { \@@_compute_i_j:nnnn #1 #2 }
8733 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8734 {
8735     \def \l_@@_first_i_tl { #1 }
8736     \def \l_@@_first_j_tl { #2 }
8737     \def \l_@@_last_i_tl { #3 }
8738     \def \l_@@_last_j_tl { #4 }
8739     \tl_if_eq:NnT \l_@@_first_i_tl { last }
8740     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8741     \tl_if_eq:NnT \l_@@_first_j_tl { last }
8742     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8743     \tl_if_eq:NnT \l_@@_last_i_tl { last }
8744     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8745     \tl_if_eq:NnT \l_@@_last_j_tl { last }
8746     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8747 }

```

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`.

```

8748 \hook_gput_code:nnn { begindocument } { . }
8749 {
8750   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
8751   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
8752     { \@@_sub_matrix:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
8753 }
8754 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
8755 {
8756   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8757 \@@_compute_i_j:nn { #2 } { #3 }
8758 \int_compare:nNt { \l_@@_first_i_tl } = { \l_@@_last_i_tl }
8759 { \def \arraystretch { 1 } }
8760 \bool_lazy_or:nnTF
8761 { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
8762 { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
8763 { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8764 {
8765   \str_clear_new:N \l_@@_submatrix_name_str
8766   \keys_set:nn { nicematrix / SubMatrix } { #5 }
8767   \pgfpicture
8768   \pgfrememberpicturepositiononpagetrue
8769   \pgf@relevantforpicturesizefalse
8770   \pgfset { inner~sep = \c_zero_dim }
8771   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8772   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currfication.

```

8773 \bool_if:NTF \l_@@_submatrix_slim_bool
8774 { \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl } }
8775 { \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int } }
8776 {
8777   \cs_if_exist:cT
8778     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8779   {
8780     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8781     \dim_compare:nNt { \pgf@x } < { \l_@@_x_initial_dim }
8782     { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8783   }
8784   \cs_if_exist:cT
8785     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8786   {
8787     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8788     \dim_compare:nNt { \pgf@x } > { \l_@@_x_final_dim }
8789     { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8790   }
8791 }
8792 \dim_compare:nNtTF { \l_@@_x_initial_dim } = { \c_max_dim }
8793 { \@@_error:nn { Impossible~delimiter } { left } }
8794 {
8795   \dim_compare:nNtTF { \l_@@_x_final_dim } = { - \c_max_dim }
8796   { \@@_error:nn { Impossible~delimiter } { right } }
8797   { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8798 }
8799 \endpgfpicture
8800 }
8801 \group_end:
8802 \ignorespaces
8803 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8804 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8805 {
8806   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8807   \dim_set:Nn \l_@@_y_initial_dim
8808   {
8809     \fp_to_dim:n
8810     {
8811       \pgf@y
8812       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8813     }
8814   }
8815   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8816   \dim_set:Nn \l_@@_y_final_dim
8817   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8818   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
8819   {
8820     \cs_if_exist:cT
8821     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8822     {
8823       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8824       \dim_set:Nn \l_@@_y_initial_dim
8825       { \dim_max:nn { \l_@@_y_initial_dim } { \pgf@y } }
8826     }
8827     \cs_if_exist:cT
8828     { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8829     {
8830       \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8831       \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
8832       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8833     }
8834   }
8835   \dim_set:Nn \l_tmpa_dim
8836   {
8837     \l_@@_y_initial_dim - \l_@@_y_final_dim +
8838     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8839   }
8840   \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the \SubMatrix.

```

8841   \group_begin:
8842   \pgfsetlinewidth { 1.1 \arrayrulewidth }
8843   \@@_set_CTarc:o \l_@@_rules_color_tl
8844   \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`.

```

8845   \seq_map_inline:Nn \g_@@_cols_vlism_seq
8846   {
8847     \int_compare:nNnT { \l_@@_first_j_tl } < { ##1 }
8848     {
8849       \int_compare:nNnT
8850       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8851       {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8852         \@@_qpoint:n { col - ##1 }
8853         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8854         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8855         \pgfusepathqstroke
8856       }
8857     }
8858   }

```

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.

```

8859 \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
8860 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8861 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8862 {
8863   \bool_lazy_and:nnTF
8864   { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
8865   {
8866     \int_compare_p:nNn
8867     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8868   {
8869     \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8870     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8871     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8872     \pgfusepathqstroke
8873   }
8874   { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8875 }

```

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.

```

8876 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
8877 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8878 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8879 {
8880   \bool_lazy_and:nnTF
8881   { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
8882   {
8883     \int_compare_p:nNn
8884     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8885   {
8886     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8887 \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8888 \dim_set:Nn \l_tmpa_dim
8889 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8890 \str_case:nn { #1 }
8891 {
8892   ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8893   [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8894   \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8895   }
8896   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8897 \dim_set:Nn \l_tmpb_dim
8898 { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8899 \str_case:nn { #2 }
8900 {
8901   ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8902   ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8903   \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8904 }
8905 \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8906 \pgfusepathqstroke
8907 \group_end:
8908 }
8909 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8910 }

```

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).

```

8911 \str_if_empty:NF \l_@@_submatrix_name_str
8912 {
8913   \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8914   \l_@@_x_initial_dim \l_@@_y_initial_dim
8915   \l_@@_x_final_dim \l_@@_y_final_dim
8916 }
8917 \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`.

```

8918 \begin { pgfscope }
8919 \pgftransformshift
8920 {
8921   \pgfpoint
8922   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8923   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8924 }
8925 \str_if_empty:NTF \l_@@_submatrix_name_str
8926 { \@@_node_left:nn #1 { } }
8927 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8928 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8929 \pgftransformshift
8930 {
8931   \pgfpoint
8932   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8933   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8934 }
8935 \str_if_empty:NTF \l_@@_submatrix_name_str
8936 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8937 {
8938   \@@_node_right:nnnn #2
8939   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8940 }

```

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`.

```

8941 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8942 \flag_clear_new:N \l_@@_code_flag
8943 \l_@@_code_tl
8944 }

```

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`.

```

8945 \cs_set_eq:NN \@@_old_pgfpointanchor: \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by currying.

```
8946 \cs_new:Npn \@@_pgfpointanchor:n #1
8947 { \exp_args:Ne \@@_old_pgfpointanchor: { \@@_pgfpointanchor_i:n { #1 } } }
```

First, we must detect whether the argument is of the form `\tikz@pp@name{...}` (the command `\tikz@pp@name` is a command of TikZ that adds the prefix and the suffix of the name. If the name refers to a TikZ node which does not exist, there isn't the wrapper `\tikz@pp@name`).

```
8948 \cs_new:Npn \@@_pgfpointanchor_i:n #1
8949 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }

8950 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
8951 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
8952 \str_if_empty:nTF { #1 }
```

First, when the name of the name begins with `\tikz@pp@name`.

```
8953 { \@@_pgfpointanchor_iv:w #2 }
```

And now, when there is no `\tikz@pp@name`.

```
8954 { \@@_pgfpointanchor_ii:n { #1 } }
8955 }
```

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`).

```
8956 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
8957 { \@@_pgfpointanchor_ii:n { #1 } }
```

With the command `\@@_pgfpointanchor_ii:n`, we deal with the actual name of the node (without the `\tikz@pp@name`). First, we have to detect whether it is of the form `i` or of the form `i-j` (with an hyphen).

Remark: It would be possible to test the presence of the hyphen in an expandable way to using `\etl_if_in:nnTF` of the package `etl` but, as of now, we do not load `etl`.

```
8958 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }
```

```
8959 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
8960 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
8961 \str_if_empty:nTF { #2 }
```

First the case where the argument does *not* contain an hyphen.

```
8962 { \@@_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`).

```
8963 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8964 }
```

The following function is for the case when the name contains an hyphen.

```
8965 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8966 {
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
8967 \@@_env:
8968 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8969 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8970 }
```

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.

```

8971 \tl_const:Nn \c_@@_integers_alist_tl
8972 {
8973   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8974   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8975   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8976   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8977 }

8978 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
8979 {

```

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`.

```

8980 \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8981 {
8982   \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

8983 \@@_env: -
8984 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8985   { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8986   { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8987 }
8988 {
8989   \str_if_eq:eeTF { #1 } { last }
8990   {
8991     \flag_raise:N \l_@@_code_flag
8992     \@@_env: -
8993     \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8994       { \int_eval:n { \l_@@_last_i_tl + 1 } }
8995       { \int_eval:n { \l_@@_last_j_tl + 1 } }
8996   }
8997   { #1 }
8998 }
8999 }

```

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`).

```

9000 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9001 {
9002   \pgfnode
9003     { rectangle }
9004     { east }
9005     {
9006       \nullfont
9007       \c_math_toggle_token
9008       \@@_color:o \l_@@_delimiters_color_tl
9009       \left #1
9010       \vcenter
9011       {
9012         \nullfont
9013         \hrule \@height \l_tmpa_dim
9014         \@depth \c_zero_dim

```



```

9015         \@width \c_zero_dim
9016     }
9017     \right .
9018     \c_math_toggle_token
9019 }
9020 { #2 }
9021 { }
9022 }

```

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.

```

9023 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9024 {
9025     \pgfnode
9026     { rectangle }
9027     { west }
9028     {
9029         \nullfont
9030         \c_math_toggle_token
9031         \colorlet { current-color } { . }
9032         \@@_color:o \l_@@_delimiters_color_tl
9033         \left .
9034         \vcenter
9035         {
9036             \nullfont
9037             \hrule \@height \l_tmpa_dim
9038                 \@depth \c_zero_dim
9039                 \@width \c_zero_dim
9040         }
9041         \right #1
9042         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9043         ^ { \color { current-color } \smash { #4 } }
9044         \c_math_toggle_token
9045     }
9046     { #2 }
9047     { }
9048 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9049 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9050 {
9051     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9052     \ignorespaces
9053 }
9054 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9055 {
9056     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9057     \ignorespaces
9058 }
9059 \keys_define:nn { nicematrix / Brace }
9060 {
9061     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9062     left-shorten .default:n = true ,
9063     left-shorten .value_forbidden:n = true ,

```

```

9064 right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9065 right-shorten .default:n = true ,
9066 right-shorten .value_forbidden:n = true ,
9067 shorten .meta:n = { left-shorten , right-shorten } ,
9068 shorten .value_forbidden:n = true ,
9069 yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9070 yshift .value_required:n = true ,
9071 yshift .initial:n = \c_zero_dim ,
9072 color .tl_set:N = \l_tmpa_tl ,
9073 color .value_required:n = true ,
9074 unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
9075 }

```

#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.

```

9076 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9077 {
9078   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9079   \@@_compute_i_j:nn { #1 } { #2 }
9080   \bool_lazy_or:nnTF
9081     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9082     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9083     {
9084       \str_if_eq:eeTF { #5 } { under }
9085       { \@@_error:nn { Construct~too~large } { \UnderBrace } }
9086       { \@@_error:nn { Construct~too~large } { \OverBrace } }
9087     }
9088     {
9089       \tl_clear:N \l_tmpa_tl
9090       \keys_set:nn { nicematrix / Brace } { #4 }
9091       \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9092       \pgfpicture
9093       \pgfrememberpicturepositiononpagetrue
9094       \pgf@relevantforpicturesizefalse
9095       \bool_if:NT \l_@@_brace_left_shorten_bool
9096       {
9097         \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9098         \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9099         {
9100           \cs_if_exist:cT
9101             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9102             {
9103               \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9104
9105               \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9106               { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9107             }
9108         }
9109       }
9110       \bool_lazy_or:nnT
9111         { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9112         { \dim_compare_p:nNn { \l_@@_x_initial_dim } = { \c_max_dim } }
9113         {
9114           \@@_qpoint:n { col - \l_@@_first_j_tl }
9115           \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9116         }
9117       \bool_if:NT \l_@@_brace_right_shorten_bool
9118       {
9119         \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9120         \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9121         {

```

```

9122         \cs_if_exist:cT
9123         { pgf @ sh @ ns @ \l_@@_env: - ##1 - \l_@@_last_j_tl }
9124         {
9125             \pgfpointanchor { \l_@@_env: - ##1 - \l_@@_last_j_tl } { east }
9126             \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
9127             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9128         }
9129     }
9130 }
9131 \bool_lazy_or:nnT
9132 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9133 { \dim_compare_p:nNn { \l_@@_x_final_dim } = { - \c_max_dim } }
9134 {
9135     \l_@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9136     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9137 }
9138 \pgfset { inner~sep = \c_zero_dim }
9139 \str_if_eq:eeTF { #5 } { under }
9140 { \l_@@_underbrace_i:n { #3 } }
9141 { \l_@@_overbrace_i:n { #3 } }
9142 \endpgfpicture
9143 }
9144 \group_end:
9145 }

```

The argument is the text to put above the brace.

```

9146 \cs_new_protected:Npn \l_@@_overbrace_i:n #1
9147 {
9148     \l_@@_qpoint:n { row - \l_@@_first_i_tl }
9149     \pgftransformshift
9150     {
9151         \pgfpoint
9152         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9153         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9154     }
9155     \pgfnode
9156     { rectangle }
9157     { south }
9158     {
9159         \vtop
9160         {
9161             \group_begin:
9162             \everycr { }
9163             \halign
9164             {
9165                 \hfil ## \hfil \crcr
9166                 \bool_if:NTF \l_@@_tabular_bool
9167                 { \begin { tabular } { c } #1 \end { tabular } }
9168                 { $ \begin { array } { c } #1 \end { array } $ }
9169                 \cr
9170                 \c_math_toggle_token
9171                 \overbrace
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             }
9180             \group_end:
9181         }
9182     }
9183 }

```

```

9184     { }
9185 }

```

The argument is the text to put under the brace.

```

9186 \cs_new_protected:Npn \@@_underbrace_i:n #1
9187 {
9188   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9189   \pgftransformshift
9190   {
9191     \pgfpoint
9192     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9193     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9194   }
9195   \pgfnode
9196   { rectangle }
9197   { north }
9198   {
9199     \group_begin:
9200     \everycr { }
9201     \vbox
9202     {
9203       \halign
9204       {
9205         \hfil ## \hfil \crcr
9206         \c_math_toggle_token
9207         \underbrace
9208         {
9209           \hbox_to_wd:nn
9210           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9211           { }
9212         }
9213         \c_math_toggle_token
9214         \cr
9215         \bool_if:NTF \l_@@_tabular_bool
9216         { \begin { tabular } { c } #1 \end { tabular } }
9217         { $ \begin { array } { c } #1 \end { array } $ }
9218         \cr
9219       }
9220     }
9221     \group_end:
9222   }
9223   { }
9224   { }
9225 }

```

35 The commands HBrace et VBrace

```

9226 \hook_gput_code:nnn { begindocument } { . }
9227 {
9228   \cs_if_exist:cT { tikz@library@decorations.pathreplacing@loaded }
9229   {
9230     \tikzset
9231     {
9232       nicematrix / brace / .style =
9233       {
9234         decoration = { brace , raise = -0.15 em } ,
9235         decorate ,
9236       } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9237         nicematrix / mirrored-brace / .style =
9238         {
9239             nicematrix / brace ,
9240             decoration = mirror ,
9241         }
9242     }
9243 }
9244 }
```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9245 \keys_define:nn { nicematrix / Hbrace }
9246 {
9247     color .code:n = ,
9248     horizontal-label .code:n = ,
9249     horizontal-labels .code:n = ,
9250     shorten .code:n = ,
9251     shorten-start .code:n = ,
9252     shorten-end .code:n = ,
9253     unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9254 }
```

Here we need an “fully expandable” command.

```

9255 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9256 {
9257     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9258     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9259     { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9260 }
```

The following command must *not* be protected.

```

9261 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9262 {
9263     \int_compare:nNnTF { \c@iRow } < { 2 }
9264     {
```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9265     \str_if_eq:nnTF { #2 } { * }
9266     {
9267         \NiceMatrixOptions { nullify-dots }
9268         \Ldots
9269         [
9270             line-style = nicematrix / brace ,
9271             #1 ,
9272             up =
9273             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9274         ]
9275     }
9276     {
9277         \Hdotsfor
9278         [
9279             line-style = nicematrix / brace ,
9280             #1 ,
9281             up =
9282             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9283         ]
9284         { #2 }
9285     }
9286 }
9287 {
9288     \str_if_eq:nnTF { #2 } { * }
9289     {
```

```

9290 \NiceMatrixOptions { nullify-dots }
9291 \Ldots
9292 [
9293   line-style = nicematrix / mirrored-brace ,
9294   #1 ,
9295   down =
9296     \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9297 ]
9298 }
9299 {
9300   \Hdotsfor
9301   [
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 \keys_set:nn { nicematrix / Hbrace } { #1 }
9311 }

```

Here we need an “fully expandable” command.

```

9312 \NewExpandableDocumentCommand { \@@_Vbrace } { 0 { } m m }
9313 {
9314   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9315     { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9316     { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9317 }

```

The following command must *not* be protected.

```

9318 \cs_new:Npn \@@_vbrace:nnn #1 #2 #3
9319 {
9320   \int_compare:nNnTF { \c@jCol } < { 2 }
9321   {
9322     \str_if_eq:nnTF { #2 } { * }
9323     {
9324       \NiceMatrixOptions { nullify-dots }
9325       \Vdots
9326       [
9327         Vbrace ,
9328         line-style = nicematrix / mirrored-brace ,
9329         #1 ,
9330         down =
9331           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9332       ]
9333     }
9334     {
9335       \Vdotsfor
9336       [
9337         Vbrace ,
9338         line-style = nicematrix / mirrored-brace ,
9339         #1 ,
9340         down =
9341           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9342       ]
9343       { #2 }
9344     }
9345   }
9346   {
9347     \str_if_eq:nnTF { #2 } { * }
9348     {
9349       \NiceMatrixOptions { nullify-dots }

```

```

9350         \Vdots
9351         [
9352             Vbrace ,
9353             line-style = nicematrix / brace ,
9354             #1 ,
9355             up =
9356             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9357         ]
9358     }
9359     {
9360         \Vdotsfor
9361         [
9362             Vbrace ,
9363             line-style = nicematrix / brace ,
9364             #1 ,
9365             up =
9366             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9367         ]
9368         { #2 }
9369     }
9370 }
9371 \keys_set:nn { nicematrix / Hbrace } { #1 }
9372 }

```

36 The command TikzEveryCell

```

9373 \bool_new:N \l_@@_not_empty_bool
9374 \bool_new:N \l_@@_empty_bool
9375
9376 \keys_define:nn { nicematrix / TikzEveryCell }
9377 {
9378     not-empty .code:n =
9379         \bool_lazy_or:nnTF
9380         { \l_@@_in_code_after_bool }
9381         { \g_@@_create_cell_nodes_bool }
9382         { \bool_set_true:N \l_@@_not_empty_bool }
9383         { \@@_error:n { detection-of-empty-cells } } } ,
9384     not-empty .value_forbidden:n = true ,
9385     empty .code:n =
9386         \bool_lazy_or:nnTF
9387         { \l_@@_in_code_after_bool }
9388         { \g_@@_create_cell_nodes_bool }
9389         { \bool_set_true:N \l_@@_empty_bool }
9390         { \@@_error:n { detection-of-empty-cells } } } ,
9391     empty .value_forbidden:n = true ,
9392     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
9393 }
9394
9395
9396 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9397 {
9398     \IfPackageLoadedTF { tikz }
9399     {
9400         \group_begin:
9401         \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.

```

9402         \tl_set:Nn \l_tmpa_tl { { #2 } }
9403         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9404         { \@@_for_a_block:nnnnn ##1 }
9405         \@@_all_the_cells:

```

```

9406     \group_end:
9407 }
9408 { \@@_error:n { TikzEveryCell~without~tikz } }
9409 }
9410
9411 \tl_new:N \l_@@_i_tl
9412 \tl_new:N \l_@@_j_tl
9413
9414
9415 \cs_new_protected:Nn \@@_all_the_cells:
9416 {
9417   \int_step_inline:nn \c@iRow
9418   {
9419     \int_step_inline:nn \c@jCol
9420     {
9421       \cs_if_exist:cF { cell - ##1 - #####1 }
9422       {
9423         \clist_if_in:NcF \l_@@_corners_cells_clist
9424         { ##1 - #####1 }
9425         {
9426           \bool_set_false:N \l_tmpa_bool
9427           \cs_if_exist:cTF
9428           { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
9429           {
9430             \bool_if:NF \l_@@_empty_bool
9431             { \bool_set_true:N \l_tmpa_bool }
9432           }
9433           {
9434             \bool_if:NF \l_@@_not_empty_bool
9435             { \bool_set_true:N \l_tmpa_bool }
9436           }
9437           \bool_if:NT \l_tmpa_bool
9438           {
9439             \@@_block_tikz:nnnnn
9440             \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
9441           }
9442         }
9443       }
9444     }
9445   }
9446 }
9447
9448 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9449 {
9450   \bool_if:NF \l_@@_empty_bool
9451   {
9452     \@@_block_tikz:nnnnn
9453     \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9454   }
9455   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9456 }
9457
9458 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9459 {
9460   \int_step_inline:nnn { #1 } { #3 }
9461   {
9462     \int_step_inline:nnn { #2 } { #4 }
9463     { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9464   }
9465 }

```


37 The command \ShowCellNames

```

9466 \NewDocumentCommand \@@_ShowCellNames { }
9467 {
9468   \bool_if:NT \l_@@_in_code_after_bool
9469   {
9470     \pgfpicture
9471     \pgfrememberpicturepositiononpagetrue
9472     \pgf@relevantforpicturesizefalse
9473     \pgfpathrectanglecorners
9474     { \@@_qpoint:n { 1 } }
9475     {
9476       \@@_qpoint:n
9477       { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
9478     }
9479     \pgfsetfillopacity { 0.75 }
9480     \pgfsetfillcolor { white }
9481     \pgfusepathqfill
9482     \endpgfpicture
9483   }
9484   \dim_gzero_new:N \g_@@_tmpc_dim
9485   \dim_gzero_new:N \g_@@_tmpd_dim
9486   \dim_gzero_new:N \g_@@_tmpe_dim
9487   \int_step_inline:nn { \c@iRow }
9488   {
9489     \bool_if:NTF \l_@@_in_code_after_bool
9490     {
9491       \pgfpicture
9492       \pgfrememberpicturepositiononpagetrue
9493       \pgf@relevantforpicturesizefalse
9494     }
9495     { \begin { pgfpicture } }
9496     \@@_qpoint:n { row - ##1 }
9497     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9498     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9499     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9500     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9501     \bool_if:NTF \l_@@_in_code_after_bool
9502     { \endpgfpicture }
9503     { \end { pgfpicture } }
9504     \int_step_inline:nn { \c@jCol }
9505     {
9506       \hbox_set:Nn \l_tmpa_box
9507       {
9508         \normalfont \Large \sffamily \bfseries
9509         \bool_if:NTF \l_@@_in_code_after_bool
9510         { \color { red } }
9511         { \color { red ! 50 } }
9512         ##1 - ####1
9513       }
9514       \bool_if:NTF \l_@@_in_code_after_bool
9515       {
9516         \pgfpicture
9517         \pgfrememberpicturepositiononpagetrue
9518         \pgf@relevantforpicturesizefalse
9519       }
9520       { \begin { pgfpicture } }
9521       \@@_qpoint:n { col - ####1 }
9522       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9523       \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9524       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9525       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x

```

```

9526 \bool_if:NTF \l_@@_in_code_after_bool
9527 { \endpgfpicture }
9528 { \end { pgfpicture } }
9529 \fp_set:Nn \l_tmpa_fp
9530 {
9531   \fp_min:nn
9532   {
9533     \fp_min:nn
9534     { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9535     { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9536   }
9537   { 1.0 }
9538 }
9539 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9540 \pgfpicture
9541 \pgfrememberpicturerepositiononpagetrue
9542 \pgf@relevantforpicturesizefalse
9543 \pgftransformshift
9544 {
9545   \pgfpoint
9546   { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9547   { \dim_use:N \g_tmpa_dim }
9548 }
9549 \pgfnode
9550 { rectangle }
9551 { center }
9552 { \box_use:N \l_tmpa_box }
9553 { }
9554 { }
9555 \endpgfpicture
9556 }
9557 }
9558 }

```

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.

```

9559 \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.

```

9560 \bool_new:N \g_@@_footnote_bool
9561 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9562 {
9563   You-have-used-the-key~' \l_keys_key_str '~when-loading-nicematrix~
9564   but-that-key-is-unknown. \\
9565   It-will-be-ignored. \\
9566   For-a-list-of-the-available-keys,~type-H~<return>.
9567 }
9568 {
9569   The-available-keys-are~(in-alphabetic-order):~
9570   footnote,~
9571   footnotehyper,~
9572   messages-for-Overleaf,~
9573   renew-dots-and~

```

```

9574     renew-matrix.
9575 }
9576 \keys_define:nn { nicematrix }
9577 {
9578     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9579     renew-dots .value_forbidden:n = true ,
9580     renew-matrix .code:n = \@@_renew_matrix: ,
9581     renew-matrix .value_forbidden:n = true ,
9582     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9583     footnote .bool_set:N = \g_@@_footnote_bool ,
9584     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9585     unknown .code:n = \@@_error:n { Unknown~key~for~package }
9586 }
9587 \ProcessKeyOptions

9588 \@@_msg_new:nn { footnote-with-footnotehyper-package }
9589 {
9590     You~can't~use~the~option~'footnote'~because~the~package~
9591     footnotehyper~has~already~been~loaded.~
9592     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9593     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9594     of~the~package~footnotehyper.\\
9595     The~package~footnote~won't~be~loaded.
9596 }
9597 \@@_msg_new:nn { footnotehyper-with-footnote-package }
9598 {
9599     You~can't~use~the~option~'footnotehyper'~because~the~package~
9600     footnote~has~already~been~loaded.~
9601     If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9602     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9603     of~the~package~footnote.\\
9604     The~package~footnotehyper~won't~be~loaded.
9605 }

9606 \bool_if:NT \g_@@_footnote_bool
9607 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9608     \IfClassLoadedTF { beamer }
9609     { \bool_set_false:N \g_@@_footnote_bool }
9610     {
9611         \IfPackageLoadedTF { footnotehyper }
9612         { \@@_error:n { footnote-with-footnotehyper~package } }
9613         { \usepackage { footnote } }
9614     }
9615 }

9616 \bool_if:NT \g_@@_footnotehyper_bool
9617 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9618     \IfClassLoadedTF { beamer }
9619     { \bool_set_false:N \g_@@_footnote_bool }
9620     {
9621         \IfPackageLoadedTF { footnote }
9622         { \@@_error:n { footnotehyper-with-footnote~package } }
9623         { \usepackage { footnotehyper } }
9624     }
9625     \bool_set_true:N \g_@@_footnote_bool
9626 }

```

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.

```

9627 \bool_new:N \l_@@_underscore_loaded_bool
9628 \IfPackageLoadedT { underscore }
9629 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9630 \hook_gput_code:nnn { begindocument } { . }
9631 {
9632   \bool_if:NF \l_@@_underscore_loaded_bool
9633   {
9634     \IfPackageLoadedT { underscore }
9635     { \@@_error:n { underscore-after-nicematrix } }
9636   }
9637 }

```

40 Error messages of the package

```

9638 \str_const:Ne \c_@@_available_keys_str
9639 {
9640   \bool_if:nTF { ! \g_@@_messages_for_Overleaf_bool }
9641   { For~a~list~of~the~available~keys,~type~H~<return>. }
9642   { }
9643 }
9644 \seq_new:N \g_@@_types_of_matrix_seq
9645 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9646 {
9647   NiceMatrix ,
9648   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9649 }
9650 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9651 { \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`.

```

9652 \cs_new_protected:Npn \@@_error_too_much_cols:
9653 {
9654   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9655   { \@@_fatal:nn { too-much-cols-for-array } }
9656   \int_compare:nNnT { \l_@@_last_col_int } = { -2 }
9657   { \@@_fatal:n { too-much-cols-for-matrix } }
9658   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
9659   { \@@_fatal:n { too-much-cols-for-matrix } }
9660   \bool_if:NF \l_@@_last_col_without_value_bool
9661   { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9662 }

```

The following command must *not* be protected since it's used in an error message.

```

9663 \cs_new:Npn \@@_message_hdotsfor:
9664 {
9665   \tl_if_empty:oF \g_@@_HVDotsfor_lines_tl
9666   { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
9667     \token_to_str:N \Hbrace \ is~incorrect. }
9668 }
9669 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9670 {
9671   Incompatible~options.\\
9672   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~the~same~time.\\
9673   The~output~will~not~be~reliable.
9674 }
9675 \@@_msg_new:nn { key~color~inside }
9676 {
9677   Key~deprecated.\\
9678   The~key~'color~inside'~(and~its~alias~'colortbl~like')~is~now~point~less~
9679   and~have~been~deprecated.\\
9680   You~won't~have~similar~message~till~the~end~of~the~document.
9681 }
9682 \@@_msg_new:nn { invalid~weight }
9683 {
9684   Unknown~key.\\
9685   The~key~' \l_keys_key_str '~of~your~column~X~is~unknown~and~will~be~ignored.
9686 }
9687 \@@_msg_new:nn { last~col~not~used }
9688 {
9689   Column~not~used.\\
9690   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
9691   in~your~\@@_full_name_env: .~
9692   However,~you~can~go~on.
9693 }
9694 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
9695 {
9696   Too~much~columns.\\
9697   In~the~row~ \int_eval:n { \c@iRow },~
9698   you~try~to~use~more~columns~
9699   than~allowed~by~your~ \@@_full_name_env: .
9700   \@@_message_hdotsfor: \
9701   The~maximal~number~of~columns~is~ \int_eval:n { \l_@@_last_col_int - 1 }~
9702   (plus~the~exterior~columns).~This~error~is~fatal.
9703 }
9704 \@@_msg_new:nn { too~much~cols~for~matrix }
9705 {
9706   Too~much~columns.\\
9707   In~the~row~ \int_eval:n { \c@iRow },~
9708   you~try~to~use~more~columns~than~allowed~by~your~ \@@_full_name_env: .
9709   \@@_message_hdotsfor: \
9710   Recall~that~the~maximal~number~of~columns~for~a~matrix~
9711   (excepted~the~potential~exterior~columns)~is~fixed~by~the~
9712   LaTeX~counter~'MaxMatrixCols'.~
9713   Its~current~value~is~ \int_use:N \c@MaxMatrixCols \
9714   (use~ \token_to_str:N \setcounter \ to~change~that~value).~
9715   This~error~is~fatal.
9716 }
9717 \@@_msg_new:nn { too~much~cols~for~array }
9718 {
9719   Too~much~columns.\\
9720   In~the~row~ \int_eval:n { \c@iRow },~
9721   ~you~try~to~use~more~columns~than~allowed~by~your~
9722   \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal~number~of~columns~is~

```

```

9723 \int_use:N \g_@@_static_num_of_col_int \
9724 \bool_if:nT
9725 { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
9726 { ~(plus~the~exterior~ones) }
9727 since~the~preamble~is~' \g_@@_user_preamble_tl '.\\
9728 This~error~is~fatal.
9729 }
9730 \@@_msg_new:nn { columns~not~used }
9731 {
9732 Columns~not~used.\\
9733 The~preamble~of~your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '~
9734 It~announces~ \int_use:N \g_@@_static_num_of_col_int \
9735 columns~but~you~only~used~ \int_use:N \c@jCol .\\
9736 The~columns~you~did~not~used~won't~be~created.\\
9737 You~won't~have~similar~warning~till~the~end~of~the~document.
9738 }
9739 \@@_msg_new:nn { empty~preamble }
9740 {
9741 Empty~preamble.\\
9742 The~preamble~of~your~ \@@_full_name_env: \ is~empty.\\
9743 This~error~is~fatal.
9744 }
9745 \@@_msg_new:nn { in~first~col }
9746 {
9747 Erroneous~use.\\
9748 You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
9749 That~command~will~be~ignored.
9750 }
9751 \@@_msg_new:nn { in~last~col }
9752 {
9753 Erroneous~use.\\
9754 You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9755 That~command~will~be~ignored.
9756 }
9757 \@@_msg_new:nn { in~first~row }
9758 {
9759 Erroneous~use.\\
9760 You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9761 That~command~will~be~ignored.
9762 }
9763 \@@_msg_new:nn { in~last~row }
9764 {
9765 Erroneous~use.\\
9766 You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9767 That~command~will~be~ignored.
9768 }
9769 \@@_msg_new:nn { TopRule~without~booktabs }
9770 {
9771 Erroneous~use.\\
9772 You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
9773 That~command~will~be~ignored.
9774 }
9775 \@@_msg_new:nn { TopRule~without~tikz }
9776 {
9777 Erroneous~use.\\
9778 You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
9779 That~command~will~be~ignored.
9780 }
9781 \@@_msg_new:nn { caption~outside~float }
9782 {

```

```

9783     Key~caption~forbidden.\\
9784     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9785     environment~(such~as~\{table\}).~This~key~will~be~ignored.
9786 }

9787 \@@_msg_new:nn { short-caption~without~caption }
9788 {
9789     You~should~not~use~the~key~'short-caption'~without~'caption'.~
9790     However,~your~'short-caption'~will~be~used~as~'caption'.
9791 }

9792 \@@_msg_new:nn { double~closing~delimiter }
9793 {
9794     Double~delimiter.\\
9795     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9796     delimiter.~This~delimiter~will~be~ignored.
9797 }

9798 \@@_msg_new:nn { delimiter~after~opening }
9799 {
9800     Double~delimiter.\\
9801     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9802     delimiter.~That~delimiter~will~be~ignored.
9803 }

9804 \@@_msg_new:nn { bad~option~for~line~style }
9805 {
9806     Bad~line~style.\\
9807     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9808     is~'standard'.~That~key~will~be~ignored.
9809 }

9810 \@@_msg_new:nn { corners~with~no~cell~nodes }
9811 {
9812     Incompatible~keys.\\
9813     You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
9814     is~in~force.\\
9815     If~you~go~on,~that~key~will~be~ignored.
9816 }

9817 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
9818 {
9819     Incompatible~keys.\\
9820     You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
9821     is~in~force.\\
9822     If~you~go~on,~those~extra~nodes~won't~be~created.
9823 }

9824 \@@_msg_new:nn { Identical~notes~in~caption }
9825 {
9826     Identical~tabular~notes.\\
9827     You~can't~put~several~notes~with~the~same~content~in~
9828     \token_to_str:N \caption \ (but~you~can~in~the~main~tabular).\\
9829     If~you~go~on,~the~output~will~probably~be~erroneous.
9830 }

9831 \@@_msg_new:nn { tabularnote~below~the~tabular }
9832 {
9833     \token_to_str:N \tabularnote \ forbidden\\
9834     You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
9835     of~your~tabular~because~the~caption~will~be~composed~below~
9836     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9837     key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .\\
9838     Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
9839     no~similar~error~will~raised~in~this~document.
9840 }

9841 \@@_msg_new:nn { Unknown~key~for~rules }
9842 {

```

```

9843     Unknown~key.\\
9844     There~is~only~two~keys~available~here:~width~and~color.\\
9845     Your~key~' \l_keys_key_str '~will~be~ignored.
9846 }

9847 \@@_msg_new:nn { Unknown~key~for~Hbrace }
9848 {
9849     Unknown~key.\\
9850     You~have~used~the~key~' \l_keys_key_str '~but~the~only~
9851     keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
9852     and~ \token_to_str:N \Vbrace \ are:~'color',~
9853     'horizontal-label(s)',~'shorten'~'shorten-end'~
9854     and~'shorten-start'.\\
9855     That~error~is~fatal.
9856 }

9857 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9858 {
9859     Unknown~key.\\
9860     There~is~only~two~keys~available~here:~
9861     'empty'~and~'not-empty'.\\
9862     Your~key~' \l_keys_key_str '~will~be~ignored.
9863 }

9864 \@@_msg_new:nn { Unknown~key~for~rotate }
9865 {
9866     Unknown~key.\\
9867     The~only~key~available~here~is~'c'.\\
9868     Your~key~' \l_keys_key_str '~will~be~ignored.
9869 }

9870 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9871 {
9872     Unknown~key.\\
9873     The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
9874     It~you~go~on,~you~will~probably~have~other~errors. \\
9875     \c_@@_available_keys_str
9876 }
9877 {
9878     The~available~keys~are~(in~alphabetic~order):~
9879     ccommand,~
9880     color,~
9881     command,~
9882     dotted,~
9883     letter,~
9884     multiplicity,~
9885     sep~color,~
9886     tikz,~and~total~width.
9887 }

9888 \@@_msg_new:nnn { Unknown~key~for~xdots }
9889 {
9890     Unknown~key.\\
9891     The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9892     \c_@@_available_keys_str
9893 }
9894 {
9895     The~available~keys~are~(in~alphabetic~order):~
9896     'color',~
9897     'horizontal(s)-labels',~
9898     'inter',~
9899     'line-style',~
9900     'radius',~
9901     'shorten',~
9902     'shorten-end'~and~'shorten-start'.
9903 }

```



```

9904 \@@_msg_new:nn { Unknown-key-for-rowcolors }
9905 {
9906   Unknown-key.\
9907   As-for-now,~there-is-only-two-keys-available-here:~'cols'~and~'respect-blocks'~
9908   (and-you-try-to-use~' \l_keys_key_str '~)\
9909   That-key-will-be-ignored.
9910 }
9911 \@@_msg_new:nn { label-without-caption }
9912 {
9913   You-can't-use-the-key~'label'~in-your~\{NiceTabular\}~because~
9914   you-have-not-used-the-key~'caption'.~The-key~'label'~will-be-ignored.
9915 }
9916 \@@_msg_new:nn { W-warning }
9917 {
9918   Line~ \msg_line_number: .~The-cell-is-too-wide-for-your-column~'W'~
9919   (row~ \int_use:N \c@iRow ).
9920 }
9921 \@@_msg_new:nn { Construct-too-large }
9922 {
9923   Construct-too-large.\
9924   Your-command~ \token_to_str:N #1
9925   can't-be-drawn-because-your-matrix-is-too-small.\
9926   That-command-will-be-ignored.
9927 }
9928 \@@_msg_new:nn { underscore-after-nicematrix }
9929 {
9930   Problem-with-'underscore'.\
9931   The-package~'underscore'~should-be-loaded-before~'nicematrix'.~
9932   You-can-go-on-but-you-won't-be-able-to-write-something-such-as:\
9933   ' \token_to_str:N \Cdots \token_to_str:N _
9934   \{ n \token_to_str:N \text \{ ~times \} \}' .
9935 }
9936 \@@_msg_new:nn { ampersand-in-light-syntax }
9937 {
9938   Ampersand-forbidden.\
9939   You-can't-use-an-ampersand~( \token_to_str:N & )~to-separate-columns-because~
9940   ~the-key~'light-syntax'~is-in-force.~This-error-is-fatal.
9941 }
9942 \@@_msg_new:nn { double-backslash-in-light-syntax }
9943 {
9944   Double-backslash-forbidden.\
9945   You-can't-use~ \token_to_str:N \
9946   ~to-separate-rows-because-the-key~'light-syntax'~
9947   is-in-force.~You-must-use-the-character~' \l_@@_end_of_row_tl '~
9948   (set-by-the-key~'end-of-row').~This-error-is-fatal.
9949 }
9950 \@@_msg_new:nn { hlines-with-color }
9951 {
9952   Incompatible-keys.\
9953   You-can't-use-the-keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9954   \token_to_str:N \Block \ when-the-key~'color'~or~'draw'~is-used.\
9955   However,~you-can-put-several-commands~ \token_to_str:N \Block.\
9956   Your-key-will-be-discarded.
9957 }
9958 \@@_msg_new:nn { bad-value-for-baseline }
9959 {
9960   Bad-value-for-baseline.\
9961   The-value-given-to~'baseline'~( \int_use:N \l_tmpa_int )~is-not~
9962   valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int\ and~
9963   \int_use:N \g_@@_row_total_int \ or-equal-to~'t',~'c'~or~'b'~or-of~

```

```

9964     the~form~'line-i'.\\
9965     A~value~of~1~will~be~used.
9966 }
9967 \@@_msg_new:nn { detection-of~empty~cells }
9968 {
9969     Problem~with~'not~empty'\\
9970     For~technical~reasons,~you~must~activate~
9971     'create~cell~nodes'~in~ \token_to_str:N \CodeBefore \
9972     in~order~to~use~the~key~' \l_keys_key_str '.\\
9973     That~key~will~be~ignored.
9974 }
9975 \@@_msg_new:nn { siunitx~not~loaded }
9976 {
9977     siunitx~not~loaded\\
9978     You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9979     That~error~is~fatal.
9980 }
9981 \@@_msg_new:nn { Invalid~name }
9982 {
9983     Invalid~name.\\
9984     You~can't~give~the~name~' \l_keys_value_tl '~to~a~ \token_to_str:N
9985     \SubMatrix \ of~your~ \@@_full_name_env: .\\
9986     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9987     This~key~will~be~ignored.
9988 }
9989 \@@_msg_new:nn { Hbrace~not~allowed }
9990 {
9991     Command~not~allowed.\\
9992     You~can't~use~the~command~ \token_to_str:N #1
9993     because~you~have~not~loaded~
9994     \IfPackageLoadedTF { tikz }
9995     { the~TikZ~library~'decorations.pathreplacing'~Use~ }
9996     { TikZ~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
9997     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\
9998     That~command~will~be~ignored.
9999 }
10000 \@@_msg_new:nn { Vbrace~not~allowed }
10001 {
10002     Command~not~allowed.\\
10003     You~can't~use~the~command~ \token_to_str:N \Vbrace \
10004     because~you~have~not~loaded~TikZ~
10005     and~the~TikZ~library~'decorations.pathreplacing'.\\
10006     Use: ~\token_to_str:N \usepackage \{tikz\}~
10007     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
10008     That~command~will~be~ignored.
10009 }
10010 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
10011 {
10012     Wrong~line.\\
10013     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
10014     \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
10015     number~is~not~valid.~It~will~be~ignored.
10016 }
10017 \@@_msg_new:nn { Impossible~delimiter }
10018 {
10019     Impossible~delimiter.\\
10020     It's~impossible~to~draw~the~#1~delimiter~of~your~
10021     \token_to_str:N \SubMatrix \ because~all~the~cells~are~empty~
10022     in~that~column.
10023     \bool_if:NT \l_@@_submatrix_slim_bool
10024     { ~Maybe~you~should~try~without~the~key~'slim'. } \\

```

```

10025     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
10026 }
10027 \@@_msg_new:nnn { width~without~X~columns }
10028 {
10029     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
10030     the~preamble~(' \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .\\
10031     That~key~will~be~ignored.
10032 }
10033 {
10034     This~message~is~the~message~'width~without~X~columns'~
10035     of~the~module~'nicematrix'.~
10036     The~experimented~users~can~disable~that~message~with~
10037     \token_to_str:N \msg_redirect_name:nnn .\\
10038 }
10039
10040 \@@_msg_new:nn { key~multiplicity~with~dotted }
10041 {
10042     Incompatible~keys. \\
10043     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
10044     in~a~'custom~line'.~They~are~incompatible. \\
10045     The~key~'multiplicity'~will~be~discarded.
10046 }
10047 \@@_msg_new:nn { empty~environment }
10048 {
10049     Empty~environment.\\
10050     Your~ \@@_full_name_env: \ is~empty.~This~error~is~fatal.
10051 }
10052 \@@_msg_new:nn { No~letter~and~no~command }
10053 {
10054     Erroneous~use.\\
10055     Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
10056     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10057     '~'command'~(to~draw~horizontal~rules).\\
10058     However,~you~can~go~on.
10059 }
10060 \@@_msg_new:nn { Forbidden~letter }
10061 {
10062     Forbidden~letter.\\
10063     You~can't~use~the~letter~'#1'~for~a~customized~line.~
10064     It~will~be~ignored.\\
10065     The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10066 }
10067 \@@_msg_new:nn { Several~letters }
10068 {
10069     Wrong~name.\\
10070     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10071     have~used~' \l_@@_letter_str ').\\
10072     It~will~be~ignored.
10073 }
10074 \@@_msg_new:nn { Delimiter~with~small }
10075 {
10076     Delimiter~forbidden.\\
10077     You~can't~put~a~delimiter~in~the~preamble~of~your~
10078     \@@_full_name_env: \
10079     because~the~key~'small'~is~in~force.\\
10080     This~error~is~fatal.
10081 }
10082 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10083 {
10084     Unknown~cell.\\

```

```

10085 Your-command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10086 the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10087 can't~be~executed~because~a~cell~doesn't~exist.\\
10088 This-command~ \token_to_str:N \line \ will~be~ignored.
10089 }
10090 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
10091 {
10092   Duplicate-name.\\
10093   The-name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10094   in~this~ \@@_full_name_env: .\\
10095   This-key~will~be~ignored.\\
10096   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10097   { For~a~list~of~the~names~already~used,~type~H~<return>. }
10098 }
10099 {
10100   The-names~already~defined~in~this~ \@@_full_name_env: \ are:~
10101   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10102 }
10103 \@@_msg_new:nn { r-or-l-with-preamble }
10104 {
10105   Erroneous-use.\\
10106   You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
10107   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10108   your~ \@@_full_name_env: .\\
10109   This-key~will~be~ignored.
10110 }
10111 \@@_msg_new:nn { Hdotsfor~in~col~0 }
10112 {
10113   Erroneous-use.\\
10114   You~can't~use~ \token_to_str:N \Hdotsfor \ in~an~exterior~column~of~
10115   the~array.~This~error~is~fatal.
10116 }
10117 \@@_msg_new:nn { bad-corner }
10118 {
10119   Bad-corner.\\
10120   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10121   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
10122   This~specification~of~corner~will~be~ignored.
10123 }
10124 \@@_msg_new:nn { bad-border }
10125 {
10126   Bad-border.\\
10127   \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
10128   (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
10129   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10130   also~use~the~key~'tikz'
10131   \IfPackageLoadedF { tikz }
10132   { ~if~you~load~the~LaTeX~package~'tikz' } ).\\
10133   This~specification~of~border~will~be~ignored.
10134 }
10135 \@@_msg_new:nn { TikzEveryCell~without~tikz }
10136 {
10137   TikZ~not~loaded.\\
10138   You~can't~use~ \token_to_str:N \TikzEveryCell \
10139   because~you~have~not~loaded~tikz.~
10140   This~command~will~be~ignored.
10141 }
10142 \@@_msg_new:nn { tikz-key~without~tikz }
10143 {
10144   TikZ~not~loaded.\\
10145   You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N

```

```

10146 \Block '~because-you-have-not-loaded-tikz.~
10147 This-key-will-be-ignored.
10148 }

10149 \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
10150 {
10151   Erroneous-use.\\
10152   In-the~ \@@_full_name_env: ,~you-must-use-the-key~
10153   'last-col'~without-value.\\
10154   However,~you-can-go-on-for-this-time~
10155   (the-value~' \l_keys_value_tl '~will-be-ignored).
10156 }

10157 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
10158 {
10159   Erroneous-use. \\
10160   In~\token_to_str:N \NiceMatrixOptions ,~you-must-use-the-key~
10161   'last-col'~without-value. \\
10162   However,~you-can-go-on-for-this-time~
10163   (the-value~' \l_keys_value_tl '~will-be-ignored).
10164 }

10165 \@@_msg_new:nn { Block-too-large-1 }
10166 {
10167   Block-too-large. \\
10168   You-try-to-draw-a-block-in-the-cell~#1-#2-of~your~matrix-but-the-matrix-is~
10169   too-small-for-that-block. \\
10170   This-block-and-maybe-others-will-be-ignored.
10171 }

10172 \@@_msg_new:nn { Block-too-large-2 }
10173 {
10174   Block-too-large. \\
10175   The-preamble-of-your~ \@@_full_name_env: \ announces~ \int_use:N
10176   \g_@@_static_num_of_col_int \
10177   columns-but-you-use-only~ \int_use:N \c_jCol \ and-that's-why-a-block~
10178   specified-in-the-cell~#1-#2-can't-be-drawn.~You-should-add-some-ampersands~
10179   (&)~at-the-end-of-the-first-row-of-your~ \@@_full_name_env: . \\
10180   This-block-and-maybe-others-will-be-ignored.
10181 }

10182 \@@_msg_new:nn { unknown-column-type }
10183 {
10184   Bad-column-type. \\
10185   The-column-type~'#1'~in-your~ \@@_full_name_env: \
10186   is-unknown. \\
10187   This-error-is-fatal.
10188 }

10189 \@@_msg_new:nn { unknown-column-type-multicolumn }
10190 {
10191   Bad-column-type. \\
10192   The-column-type~'#1'~in-the-command~\token_to_str:N \multicolumn \
10193   ~of-your~ \@@_full_name_env: \
10194   is-unknown. \\
10195   This-error-is-fatal.
10196 }

10197 \@@_msg_new:nn { unknown-column-type-S }
10198 {
10199   Bad-column-type. \\
10200   The-column-type~'S'~in-your~ \@@_full_name_env: \ is-unknown. \\
10201   If-you-want-to-use-the-column-type~'S'~of-siunitx,~you-should~
10202   load-that-package. \\
10203   This-error-is-fatal.
10204 }

10205 \@@_msg_new:nn { unknown-column-type-S-multicolumn }

```

```

10206 {
10207   Bad-column-type. \\
10208   The-column-type-'S'~in-the-command~\token_to_str:N \multicolumn \
10209   of~your~ @@_full_name_env: \ is-unknown. \\
10210   If-you-want-to-use-the-column-type-'S'~of-siunitx,~you-should-
10211   load-that-package. \\
10212   This-error-is-fatal.
10213 }
10214 \@@_msg_new:nn { tabularnote~forbidden }
10215 {
10216   Forbidden-command. \\
10217   You-can't-use-the-command~ \token_to_str:N \tabularnote \
10218   ~here.~This-command-is-available-only-in~
10219   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
10220   the~argument~of~a~command~\token_to_str:N \caption \ included-
10221   in~an~environment~\{table\}. \\
10222   This-command-will-be-ignored.
10223 }
10224 \@@_msg_new:nn { borders~forbidden }
10225 {
10226   Forbidden-key.\\
10227   You-can't-use-the-key~'borders'~of-the-command~ \token_to_str:N \Block \
10228   because-the-option~'rounded-corners'~
10229   is~in~force~with~a~non-zero~value.\\
10230   This-key-will-be-ignored.
10231 }
10232 \@@_msg_new:nn { bottomrule~without~booktabs }
10233 {
10234   booktabs~not~loaded.\\
10235   You-can't-use-the-key~'tabular/bottomrule'~because-you-haven't~
10236   loaded~'booktabs'.\\
10237   This-key-will-be-ignored.
10238 }
10239 \@@_msg_new:nn { enumitem~not~loaded }
10240 {
10241   enumitem~not~loaded. \\
10242   You-can't-use-the-command~ \token_to_str:N \tabularnote \
10243   ~because-you-haven't~loaded~'enumitem'. \\
10244   All-the-commands~ \token_to_str:N \tabularnote \ will-be-
10245   ignored-in-the-document.
10246 }
10247 \@@_msg_new:nn { tikz~without~tikz }
10248 {
10249   Tikz~not~loaded. \\
10250   You-can't-use-the-key~'tikz'~here~because~Tikz~is~not~
10251   loaded.~If-you-go-on,~that~key~will~be~ignored.
10252 }
10253 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
10254 {
10255   Tikz~not~loaded. \\
10256   You-have-used-the-key~'tikz'~in-the-definition-of~a~
10257   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
10258   You-can-go-on~but~you~will~have~another~error~if~you~actually~
10259   use~that~custom~line.
10260 }
10261 \@@_msg_new:nn { tikz~in~borders~without~tikz }
10262 {
10263   Tikz~not~loaded. \\
10264   You-have-used-the-key~'tikz'~in~a~key~'borders'~(of~a~
10265   command~' \token_to_str:N \Block ')~but~tikz~is~not~loaded.~
10266   That~key~will~be~ignored.

```

```

10267 }
10268 \@@_msg_new:nn { color~in~custom~line~with~tikz }
10269 {
10270   Erroneous~use.\
10271   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
10272   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
10273   The~key~'color'~will~be~discarded.
10274 }
10275 \@@_msg_new:nn { Wrong~last~row }
10276 {
10277   Wrong~number.\
10278   You~have~used~'last~row= \int_use:N \l_@@_last_row_int '~but~your~
10279   \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
10280   If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
10281   last~row.~You~can~avoid~this~problem~by~using~'last~row'~
10282   without~value~(more~compilations~might~be~necessary).
10283 }
10284 \@@_msg_new:nn { Yet~in~env }
10285 {
10286   Nested~environments.\
10287   Environments~of~nicematrix~can't~be~nested.\
10288   This~error~is~fatal.
10289 }
10290 \@@_msg_new:nn { Outside~math~mode }
10291 {
10292   Outside~math~mode.\
10293   The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
10294   (and~not~in~ \token_to_str:N \vcenter ).\
10295   This~error~is~fatal.
10296 }
10297 \@@_msg_new:nn { One~letter~allowed }
10298 {
10299   Bad~name.\
10300   The~value~of~key~' \l_keys_key_str '~must~be~of~length~1~and~
10301   you~have~used~' \l_keys_value_tl '~.\
10302   It~will~be~ignored.
10303 }
10304 \@@_msg_new:nn { TabularNote~in~CodeAfter }
10305 {
10306   Environment~\{TabularNote\}~forbidden.\
10307   You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
10308   but~*before*~the~ \token_to_str:N \CodeAfter . \
10309   This~environment~\{TabularNote\}~will~be~ignored.
10310 }
10311 \@@_msg_new:nn { varwidth~not~loaded }
10312 {
10313   varwidth~not~loaded.\
10314   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
10315   loaded.\
10316   Your~column~will~behave~like~'p'.
10317 }
10318 \@@_msg_new:nn { varwidth~not~loaded~in~X }
10319 {
10320   varwidth~not~loaded.\
10321   You~can't~use~the~key~'V'~in~your~column~'X'~
10322   because~'varwidth'~is~not~loaded.\
10323   It~will~be~ignored. \
10324 }
10325 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
10326 {

```

```

10327     Unknown~key.\\
10328     Your~key~' \l_keys_key_str '~is~unknown~for~a~rule.\\
10329     \c_@@_available_keys_str
10330 }
10331 {
10332     The~available~keys~are~(in~alphabetic~order):~
10333     color,~
10334     dotted,~
10335     multiplicity,~
10336     sep-color,~
10337     tikz,~and~total-width.
10338 }
10339
10340 \@@_msg_new:nnn { Unknown~key~for~Block }
10341 {
10342     Unknown~key. \\
10343     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10344     \token_to_str:N \Block . \\
10345     It~will~be~ignored. \\
10346     \c_@@_available_keys_str
10347 }
10348 {
10349     The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10350     b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
10351     opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
10352     and~vlines.
10353 }
10354 \@@_msg_new:nnn { Unknown~key~for~Brace }
10355 {
10356     Unknown~key.\\
10357     The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
10358     \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
10359     It~will~be~ignored. \\
10360     \c_@@_available_keys_str
10361 }
10362 {
10363     The~available~keys~are~(in~alphabetic~order):~color,~left-shorten,~
10364     right-shorten,~shorten~(which~fixes~both~left-shorten~and~
10365     right-shorten)~and~yshift.
10366 }
10367 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10368 {
10369     Unknown~key.\\
10370     The~key~' \l_keys_key_str '~is~unknown.\\
10371     It~will~be~ignored. \\
10372     \c_@@_available_keys_str
10373 }
10374 {
10375     The~available~keys~are~(in~alphabetic~order):~
10376     delimiters/color,~
10377     rules~(with~the~subkeys~'color'~and~'width'),~
10378     sub-matrix~(several~subkeys)~
10379     and~xdots~(several~subkeys).~
10380     The~latter~is~for~the~command~ \token_to_str:N \line .
10381 }
10382 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10383 {
10384     Unknown~key.\\
10385     The~key~' \l_keys_key_str '~is~unknown.\\
10386     It~will~be~ignored. \\
10387     \c_@@_available_keys_str
10388 }

```



```

10389 {
10390     The~available~keys~are~(in~alphabetic~order):~
10391     create~cell~nodes,~
10392     delimiters/color~and~
10393     sub~matrix~(several~subkeys).
10394 }
10395 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10396 {
10397     Unknown~key.\\
10398     The~key~' \l_keys_key_str '~is~unknown.\\
10399     That~key~will~be~ignored. \\
10400     \c_@@_available_keys_str
10401 }
10402 {
10403     The~available~keys~are~(in~alphabetic~order):~
10404     'delimiters/color',~
10405     'extra~height',~
10406     'hlines',~
10407     'hvlines',~
10408     'left~xshift',~
10409     'name',~
10410     'right~xshift',~
10411     'rules'~(with~the~subkeys~'color'~and~'width'),~
10412     'slim',~
10413     'vlines'~and~'xshift'~(which~sets~both~'left~xshift'~
10414     and~'right~xshift').\\
10415 }
10416 \@@_msg_new:nnn { Unknown~key~for~notes }
10417 {
10418     Unknown~key.\\
10419     The~key~' \l_keys_key_str '~is~unknown.\\
10420     That~key~will~be~ignored. \\
10421     \c_@@_available_keys_str
10422 }
10423 {
10424     The~available~keys~are~(in~alphabetic~order):~
10425     bottomrule,~
10426     code~after,~
10427     code~before,~
10428     detect~duplicates,~
10429     enumitem~keys,~
10430     enumitem~keys~para,~
10431     para,~
10432     label~in~list,~
10433     label~in~tabular~and~
10434     style.
10435 }
10436 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10437 {
10438     Unknown~key.\\
10439     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10440     \token_to_str:N \RowStyle . \\
10441     That~key~will~be~ignored. \\
10442     \c_@@_available_keys_str
10443 }
10444 {
10445     The~available~keys~are~(in~alphabetic~order):~
10446     bold,~
10447     cell~space~top~limit,~
10448     cell~space~bottom~limit,~
10449     cell~space~limits,~
10450     color,~
10451     fill~(alias:~rowcolor),~

```

```

10452     nb-rows,~
10453     opacity~and~
10454     rounded-corners.
10455 }
10456 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10457 {
10458     Unknown~key.\\
10459     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10460     \token_to_str:N \NiceMatrixOptions . \\
10461     That~key~will~be~ignored. \\
10462     \c_@@_available_keys_str
10463 }
10464 {
10465     The~available~keys~are~(in~alphabetic~order):~
10466     &~in~blocks,~
10467     allow~duplicate~names,~
10468     ampersand~in~blocks,~
10469     caption~above,~
10470     cell~space~bottom~limit,~
10471     cell~space~limits,~
10472     cell~space~top~limit,~
10473     code~for~first~col,~
10474     code~for~first~row,~
10475     code~for~last~col,~
10476     code~for~last~row,~
10477     corners,~
10478     custom~key,~
10479     create~extra~nodes,~
10480     create~medium~nodes,~
10481     create~large~nodes,~
10482     custom~line,~
10483     delimiters~(several~subkeys),~
10484     end~of~row,~
10485     first~col,~
10486     first~row,~
10487     hlines,~
10488     hvlines,~
10489     hvlines~except~borders,~
10490     last~col,~
10491     last~row,~
10492     left~margin,~
10493     light~syntax,~
10494     light~syntax~expanded,~
10495     matrix/columns~type,~
10496     no~cell~nodes,~
10497     notes~(several~subkeys),~
10498     nullify~dots,~
10499     pgf~node~code,~
10500     renew~dots,~
10501     renew~matrix,~
10502     respect~arraystretch,~
10503     rounded~corners,~
10504     right~margin,~
10505     rules~(with~the~subkeys~'color'~and~'width'),~
10506     small,~
10507     sub~matrix~(several~subkeys),~
10508     vl~lines,~
10509     xdots~(several~subkeys).
10510 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

10511 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10512 {

```

```

10513     Unknown~key.\\
10514     The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
10515     \{NiceArray\}. \\
10516     That~key~will~be~ignored. \\
10517     \c_@@_available_keys_str
10518 }
10519 {
10520     The~available~keys~are~(in~alphabetic~order):~
10521     &~in~blocks,~
10522     ampersand~in~blocks,~
10523     b,~
10524     baseline,~
10525     c,~
10526     cell~space~bottom~limit,~
10527     cell~space~limits,~
10528     cell~space~top~limit,~
10529     code~after,~
10530     code~for~first~col,~
10531     code~for~first~row,~
10532     code~for~last~col,~
10533     code~for~last~row,~
10534     columns~width,~
10535     corners,~
10536     create~extra~nodes,~
10537     create~medium~nodes,~
10538     create~large~nodes,~
10539     extra~left~margin,~
10540     extra~right~margin,~
10541     first~col,~
10542     first~row,~
10543     hlines,~
10544     hvlines,~
10545     hvlines~except~borders,~
10546     last~col,~
10547     last~row,~
10548     left~margin,~
10549     light~syntax,~
10550     light~syntax~expanded,~
10551     name,~
10552     no~cell~nodes,~
10553     nullify~dots,~
10554     pgf~node~code,~
10555     renew~dots,~
10556     respect~arraystretch,~
10557     right~margin,~
10558     rounded~corners,~
10559     rules~(with~the~subkeys~'color'~and~'width'),~
10560     small,~
10561     t,~
10562     vlines,~
10563     xdots/color,~
10564     xdots/shorten~start,~
10565     xdots/shorten~end,~
10566     xdots/shorten~and~
10567     xdots/line~style.
10568 }

```

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`).

```

10569 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10570 {
10571     Unknown~key.\\
10572     The~key~' \l_keys_key_str '~is~unknown~for~the~
10573     \@@_full_name_env: . \\

```

```

10574     That~key~will~be~ignored. \\
10575     \c_@@_available_keys_str
10576 }
10577 {
10578     The~available~keys~are~(in~alphabetic~order):~
10579     &~in~blocks,~
10580     ampersand~in~blocks,~
10581     b,~
10582     baseline,~
10583     c,~
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~type,~
10593     columns~width,~
10594     corners,~
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     l,~
10606     last~col,~
10607     last~row,~
10608     left~margin,~
10609     light~syntax,~
10610     light~syntax~expanded,~
10611     name,~
10612     no~cell~nodes,~
10613     nullify~dots,~
10614     pgf~node~code,~
10615     r,~
10616     renew~dots,~
10617     respect~arraystretch,~
10618     right~margin,~
10619     rounded~corners,~
10620     rules~(with~the~subkeys~'color'~and~'width'),~
10621     small,~
10622     t,~
10623     vlines,~
10624     xdots/color,~
10625     xdots/shorten~start,~
10626     xdots/shorten~end,~
10627     xdots/shorten~and~
10628     xdots/line~style.
10629 }
10630 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10631 {
10632     Unknown~key.\\
10633     The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
10634     \{NiceTabular\}. \\
10635     That~key~will~be~ignored. \\
10636     \c_@@_available_keys_str

```

```

10637 }
10638 {
10639   The-available-keys-are~(in~alphabetic~order):~
10640   &-in-blocks,~
10641   ampersand-in-blocks,~
10642   b,~
10643   baseline,~
10644   c,~
10645   caption,~
10646   cell-space-bottom-limit,~
10647   cell-space-limits,~
10648   cell-space-top-limit,~
10649   code-after,~
10650   code-for-first-col,~
10651   code-for-first-row,~
10652   code-for-last-col,~
10653   code-for-last-row,~
10654   columns-width,~
10655   corners,~
10656   custom-line,~
10657   create-extra-nodes,~
10658   create-medium-nodes,~
10659   create-large-nodes,~
10660   extra-left-margin,~
10661   extra-right-margin,~
10662   first-col,~
10663   first-row,~
10664   hlines,~
10665   hvlines,~
10666   hvlines-except-borders,~
10667   label,~
10668   last-col,~
10669   last-row,~
10670   left-margin,~
10671   light-syntax,~
10672   light-syntax-expanded,~
10673   name,~
10674   no-cell-nodes,~
10675   notes~(several~subkeys),~
10676   nullify-dots,~
10677   pgf-node-code,~
10678   renew-dots,~
10679   respect-arraystretch,~
10680   right-margin,~
10681   rounded-corners,~
10682   rules~(with~the~subkeys~'color'~and~'width'),~
10683   short-caption,~
10684   t,~
10685   tabularnote,~
10686   vlines,~
10687   xdots/color,~
10688   xdots/shorten-start,~
10689   xdots/shorten-end,~
10690   xdots/shorten-and~
10691   xdots/line-style.
10692 }
10693 \@@_msg_new:nnn { Duplicate-name }
10694 {
10695   Duplicate-name.\\
10696   The-name-' \l_keys_value_tl 'is-already-used-and-you-shouldn't-use~
10697   the-same-environment-name-twice.~You-can-go-on,~but,~
10698   maybe,~you-will-have-incorrect-results-especially~
10699   if~you-use~'columns-width=auto'.~If~you~don't~want~to~see~this~

```

```

10700     message~again,~use~the~key~'allow~duplicate~names'~in~
10701     ' \token_to_str:N \NiceMatrixOptions '.\\
10702     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10703     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10704 }
10705 {
10706     The~names~already~defined~in~this~document~are:~
10707     \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
10708 }
10709 \@@_msg_new:nn { Option~auto~for~columns~width }
10710 {
10711     Erroneous~use.\\
10712     You~can't~give~the~value~'auto'~to~the~key~'columns~width'~here.~
10713     That~key~will~be~ignored.
10714 }
10715 \@@_msg_new:nn { NiceTabularX~without~X }
10716 {
10717     NiceTabularX~without~X.\\
10718     You~should~not~use~\{NiceTabularX\}~without~X~columns.\\
10719     However,~you~can~go~on.
10720 }
10721 \@@_msg_new:nn { Preamble~forgotten }
10722 {
10723     Preamble~forgotten.\\
10724     You~have~probably~forgotten~the~preamble~of~your~
10725     \@@_full_name_env: . \\
10726     This~error~is~fatal.
10727 }
10728 \@@_msg_new:nn { Invalid~col~number }
10729 {
10730     Invalid~column~number.\\
10731     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
10732     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.
10733 }
10734 \@@_msg_new:nn { Invalid~row~number }
10735 {
10736     Invalid~row~number.\\
10737     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
10738     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.
10739 }
10740 \@@_define_com:NNN p ( )
10741 \@@_define_com:NNN b [ ]
10742 \@@_define_com:NNN v | |
10743 \@@_define_com:NNN V \| \|
10744 \@@_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>	76
13	The environment <code>{NiceMatrix}</code> and its variants	93
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	94
15	After the construction of the array	96
16	We draw the dotted lines	102
17	The actual instructions for drawing the dotted lines with Tikz	117
18	User commands available in the new environments	123
19	The command <code>\line</code> accessible in code-after	129
20	The command <code>\RowStyle</code>	130
21	Colors of cells, rows and columns	133
22	The vertical and horizontal rules	145
23	The empty corners	162
24	The environment <code>{NiceMatrixBlock}</code>	164
25	The extra nodes	165
26	The blocks	170
27	How to draw the dotted lines transparently	195
28	Automatic arrays	195
29	The redefinition of the command <code>\dotfill</code>	196
30	The command <code>\diagbox</code>	197

31	The keyword <code>\CodeAfter</code>	198
32	The delimiters in the preamble	199
33	The command <code>\SubMatrix</code>	200
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	209
35	The commands <code>HBrace</code> et <code>VBrace</code>	212
36	The command <code>TikzEveryCell</code>	215
37	The command <code>\ShowCellNames</code>	217
38	We process the options at package loading	218
39	About the package underscore	220
40	Error messages of the package	220