

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

November 23, 2025

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

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

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

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

*This document corresponds to the version 7.4b of `nicematrix`, at the date of 2025/11/23.

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

```

20 \RequirePackage { amsmath }

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

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

```

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

```

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

```

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

```

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

```

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

```

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

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

53 \hook_gput_code:nnn { begindocument / end } { . }
54 { \IfPackageLoadedT { mdwtab } { \@@_fatal:n { mdwtab-loaded } } }

```

2 Collecting options

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

Example :

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

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

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

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

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

```

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

```

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

```

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

```

3 Technical definitions

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

```

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

```

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

```

81 \tl_new:N \l_@@_argspec_tl

```

```

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

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

```

169         }
170     }

```

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

```

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

```

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

```

176     }
177 }
178 {
179     \cs_new_protected:Npn \@@_revert_colortbl: { }
180     \cs_new_protected:Npn \@@_replace_columncolor:
181     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }

```

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

```

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

```

Idem for `\CT@drs@`.

```

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

```

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

```

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

```

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

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

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

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

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

```
223 \cs_set:Npn \@@_cline:
```

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

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

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

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

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

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

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

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

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

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

¹See question 99041 on TeX StackExchange.

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```


4 Parameters

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
306 \dim_new:N \l_@@_col_width_dim
307 \dim_set:Nn \l_@@_col_width_dim { -1 cm }
```

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

```
308 \int_new:N \g_@@_row_total_int
309 \int_new:N \g_@@_col_total_int
```

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

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

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

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

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

```
312 \tl_new:N \l_@@_hpos_cell_tl
313 \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
```

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

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

Idem for the mono-row blocks.

```
315 \dim_new:N \g_@@_blocks_ht_dim
316 \dim_new:N \g_@@_blocks_dp_dim
```

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

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

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

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

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

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

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

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

```

322 \bool_new:N \l_@@_initial_open_bool
323 \bool_new:N \l_@@_final_open_bool
324 \bool_new:N \l_@@_Vbrace_bool

```

If the user uses `{NiceTabular*}`, the width of the tabular (in the first argument of the environment `{NiceTabular*}`) will be stored in the following dimension.

```

325 \dim_new:N \l_@@_tabular_width_dim

```

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

```

326 \dim_new:N \l_@@_rule_width_dim

```

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

```

327 \tl_new:N \l_@@_rule_color_tl

```

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

```

328 \bool_new:N \g_@@_rotate_bool

```

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

```

329 \bool_new:N \g_@@_rotate_c_bool

```

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

```

330 \bool_new:N \l_@@_X_bool

```

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

```

331 \bool_new:N \l_@@_V_of_X_bool

```

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

```

332 \bool_new:N \g_@@_V_of_X_bool

```

```

333 \bool_new:N \g_@@_caption_finished_bool

```

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

```

334 \bool_new:N \l_@@_no_cell_nodes_bool

```

We will write in `\g_@@_aux_tl` all the instructions that we have to write on the `aux` file for the current environment. The content of that token list will be written on the `aux` file at the end of the environment (in an instruction `\tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }`).

```

335 \tl_new:N \g_@@_aux_tl

```

During the second run, if information concerning the current environment has been found in the `aux` file, the following flag will be raised.

```

336 \bool_new:N \g_@@_aux_found_bool

```

In particular, in that `aux` file, there will be, for each environment of `nicematrix`, an affectation for the the following sequence that will contain information about the size of the array.

```

337 \seq_new:N \g_@@_size_seq

```

```

338 \tl_new:N \g_@@_left_delim_tl

```

```

339 \tl_new:N \g_@@_right_delim_tl

```

The token list `\g_@@_user_preamble_tl` will contain the preamble provided by the the final user of `nicematrix` (eg the preamble of an environment `{NiceTabular}`).

```
340 \tl_new:N \g_@@_user_preamble_tl
```

The token list `\g_@@_array_preamble_tl` will contain the preamble constructed by `nicematrix` for the environment `{array}` (of `array`).

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

For `\multicolumn`.

```
342 \tl_new:N \g_@@_preamble_tl
```

The following parameter corresponds to the key `columns-type` of the environments `{NiceMatrix}`, `{pNiceMatrix}`, etc. and also the key `matrix / columns-type` of `\NiceMatrixOptions`.

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

```
344 \str_set:Nn \l_@@_columns_type_tl { c }
```

The following parameters correspond to the keys `down`, `up` and `middle` of a command such as `\Cdots`. Usually, the final user doesn't use that keys directly because he uses the syntax with the embellishments `_`, `^` and `..`.

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

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

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

We will store in the following sequence information provided by the instructions `\rowlistcolors` in the main array (not in the `\CodeBefore`).

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

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

```
350 {
```

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

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

```
353   \fi:
```

```
354 }
```

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

```
355 \seq_new:N \g_@@_cols_vlism_seq
```

The following colors will be used to memorize the color of the potential “first col” and the potential “first row”.

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

```
357 \colorlet { nicematrix-last-row } { . }
```

The following string is the name of the current environment or the current command of `nicematrix` (despite its name which contains `env`).

```
358 \str_new:N \g_@@_name_env_str
```

The following string will contain the word *command* or *environment* whether we are in a command of `nicematrix` or in an environment of `nicematrix`. The default value is *environment*.

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

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

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

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

```

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

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

```

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

```

369 \tl_new:N \l_@@_code_tl

```

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

```

370 \tl_new:N \l_@@_pgf_node_code_tl

```

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

```

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

```

The value of the key `code-before` will be added to the left of `\g_@@_pre_code_before_tl`. Idem for the code between `\CodeBefore` and `\Body`.

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

```

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

```

The `\CodeAfter` provided by the final user (with the key `code-after` or the keyword `\CodeAfter`) will be stored in the second token list.

```

375 \bool_new:N \l_@@_in_code_after_bool

```

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

```

376 \bool_new:N \l_@@_ampersand_bool

```

The counters `\l_@@_old_iRow_int` and `\l_@@_old_jCol_int` will be used to save the values of the potential LaTeX counters `iRow` and `jCol`. These LaTeX counters will be restored at the end of the environment.

```

377 \int_new:N \l_@@_old_iRow_int
378 \int_new:N \l_@@_old_jCol_int

```

The TeX counters `\c@iRow` and `\c@jCol` will be created in the beginning of `{NiceArrayWithDelims}` (if they don't exist previously).

The following sequence will contain the names (without backslash) of the commands created by `custom-line` by the key `command` or `ccommand` (commands used by the final user in order to draw horizontal rules).

```

379 \seq_new:N \l_@@_custom_line_commands_seq

```

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

```

380 \tl_new:N \l_@@_rules_color_tl

```

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

```
381 \fp_new:N \g_@@_total_X_weight_fp
```

If there is at least one X-column in the preamble of the array, the following flag will be raised via the `aux` file. The length `l_@@_x_columns_dim` will be the width of X-columns of weight 1.0 (the width of a column of weight x will be that dimension multiplied by x). That value is computed after the construction of the array during the first compilation in order to be used in the following run.

```
382 \bool_new:N \l_@@_X_columns_aux_bool
383 \dim_new:N \l_@@_X_columns_dim
```

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

```
384 \bool_new:N \g_@@_after_col_zero_bool
```

A kind of false row will be inserted at the end of the array for the construction of the `col` nodes (and also to fix the width of the columns when `columns-width` is used). When this special row will be created, we will raise the flag `\g_@@_row_of_col_done_bool` in order to avoid some actions set in the redefinition of `\everycr` when the last `\cr` of the `\halign` will occur (after that row of `col` nodes).

```
385 \bool_new:N \g_@@_row_of_col_done_bool
```

It's possible to use the command `\NotEmpty` to specify explicitly that a cell must be considered as non empty by `nicematrix` (the Tikz nodes are constructed only in the non empty cells).

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

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

```
388 \bool_new:N \l_@@_code_before_bool
```

The following token list will contain the code inserted in each cell of the current row (this token list will be cleared at the beginning of each row).

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

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

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

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

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

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

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

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

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

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

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

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

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

```
400 \bool_new:N \g_@@_empty_cell_bool
```

The following dimensions will be used internally to compute the width of the potential “first column” and “last column”.

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

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

The following sequence will contain the characteristics of the blocks of the array, specified by the command `\Block`. Each block is represented by 6 components surrounded by curly braces: `{imin}{jmin}{imax}{jmax}{options}{contents}`.

The variable is global because it will be modified in the cells of the array.

```
403 \seq_new:N \g_@@_blocks_seq
```

We also manage a sequence of the *positions* of the blocks. In that sequence, each block is represented by only five components: `{imin}{jmin}{imax}{jmax}{ name}`. A block with the key `hvlines` won't appear in that sequence (otherwise, the lines in that block would not be drawn!).

```
404 \seq_new:N \g_@@_pos_of_blocks_seq
```

In fact, this sequence will also contain the positions of the cells with a `\diagbox`. The sequence `\g_@@_pos_of_blocks_seq` will be used when we will draw the rules (which respect the blocks).

In the `\CodeBefore`, the value of `\g_@@_pos_of_blocks_seq` will be the value read in the `aux` file from a previous run. However, in the `\CodeBefore`, the commands `\EmptyColumn` and `\EmptyRow` will write virtual positions of blocks in the following sequence.

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

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

```
406 \seq_new:N \g_@@_pos_of_xdots_seq
```

The sequence `\g_@@_pos_of_xdots_seq` will be used when we will draw the rules required by the key `hvlines` (these rules won't be drawn within the virtual blocks corresponding to the dotted lines).

The final user may decide to “stroke” a block (using, for example, the key `draw=red!15` when using the command `\Block`). In that case, the rules specified, for instance, by `hvlines` must not be drawn around the block. That's why we keep the information of all that stroken blocks in the following sequence.

```
407 \seq_new:N \g_@@_pos_of_stroken_blocks_seq
```

If the user has used the key `corners`, all the cells which are in an (empty) corner will be stored in the following list. We use a `clist` instead of a `seq` because we will frequently search in that list (and searching in a `clist` is faster than searching in a `seq`).

```
408 \clist_new:N \l_@@_corners_cells_clist
```

The list of the names of the potential `\SubMatrix` in the `\CodeAfter` of an environment. Unfortunately, that list has to be global (we have to use it inside the group for the options of a given `\SubMatrix`).

```
409 \seq_new:N \g_@@_submatrix_names_seq
```

The following flag will be raised if the key `width` is used in an environment `{NiceTabular}` (not in a command `\NiceMatrixOptions`). You use it to raise an error when this key is used while no column `X` is used.

```
410 \bool_new:N \l_@@_width_used_bool
```

The sequence `\g_@@_multicolumn_cells_seq` will contain the list of the cells of the array where a command `\multicolumn{n}{...}{...}` with $n > 1$ is issued. In `\g_@@_multicolumn_sizes_seq`, the “sizes” (that is to say the values of n) correspondent will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

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

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

By default, the diagonal lines will be 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.

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

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

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

The following counters will be used when searching the extremities of a dotted line (we need these counters because of the potential “open” lines in the `\SubMatrix`—the `\SubMatrix` in the code-before).

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

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

The following counters will be used when drawing the rules.

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

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

```
432 \seq_new:N \g_@@_submatrix_seq
```

We are able to determine the number of columns specified in the preamble (for the environments with explicit preamble of course and without the potential exterior columns).

```
433 \int_new:N \g_@@_static_num_of_col_int
```

The following parameters correspond to the keys `fill`, `opacity`, `draw`, `tikz`, `borders`, and `rounded-corners` of the command `\Block`.

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

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

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

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

```
440 \dim_new:N \l_@@_tab_rounded_corners_dim
```

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

```
441 \tl_new:N \l_@@_color_tl
```

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

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

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

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

The parameters of the horizontal position of the label of a block. If the user uses the key `c` or `C`, the value is `c`. If the user uses the key `l` or `L`, the value is `l`. If the user uses the key `r` or `R`, the value is `r`. If the user has used a capital letter, the boolean `\l_@@_hpos_of_block_cap_bool` will be raised (in the second pass of the analyze of the keys of the command `\Block`).

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

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

```
448 \bool_new:N \l_@@_nocolor_used_bool
```

For the vertical position, the possible values are `c`, `t`, `b`, `T` and `B` (but `\l_@@_vpos_block_str` will remain empty if the user doesn’t use a key for the vertical position).

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

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

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

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

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

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

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

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

The following key is set when the keys `hvlines` and `hvlines-except-borders` are used. It’s used only to change slightly the clipping path set by the key `rounded-corners` (for a `{tabular}`).

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

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

```
462 \bool_new:N \l_@@_dotted_bool
```

The following flag will be set to true during the composition of a caption specified (by the key `caption`).

```
463 \bool_new:N \l_@@_in_caption_bool
```

Variables for the exterior rows and columns

The keys for the exterior rows and columns are `first-row`, `first-col`, `last-row` and `last-col`. However, internally, these keys are not coded in a similar way.

• First row

The integer `\l_@@_first_row_int` is the number of the first row of the array. The default value is 1, but, if the option `first-row` is used, the value will be 0.

```
464 \int_new:N \l_@@_first_row_int
465 \int_set_eq:NN \l_@@_first_row_int \c_one_int
```

• First column

The integer `\l_@@_first_col_int` is the number of the first column of the array. The default value is 1, but, if the option `first-col` is used, the value will be 0.

```
466 \int_new:N \l_@@_first_col_int
467 \int_set_eq:NN \l_@@_first_col_int \c_one_int
```

• Last row

The counter `\l_@@_last_row_int` is the number of the potential “last row”, as specified by the key `last-row`. A value of `-2` means that there is no “last row”. A value of `-1` means that there is a “last row” but we don’t know the number of that row (the key `last-row` has been used without value and the actual value has not still been read in the `aux` file).

```
468 \int_new:N \l_@@_last_row_int
469 \int_set:Nn \l_@@_last_row_int { -2 }
```

If, in an environment like `{pNiceArray}`, the option `last-row` is used without value, we will globally raise the following flag. It will be used to know if we have, after the construction of the array, to write in the `aux` file the number of the “last row”.³

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

Idem for `\l_@@_last_col_without_value_bool`

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

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.

- **Last column**

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

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

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

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

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

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

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

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

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

Some utilities

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

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

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

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

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

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

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

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

```

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

```

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

```

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

```

The following internal parameters are for:

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

```

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

```

5 The command `\tabularnote`

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

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

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

- During the composition of the caption (value of `\l_@@_caption_tl`), the tabular notes will be numbered from 1 to `\g_@@_notes_caption_int` and the notes themselves will be stored in `\g_@@_notes_in_caption_seq`. The structure of the components of that sequence will be the same as for `\g_@@_notes_seq`.
- After the composition of the main tabular and after the composition of the caption, the sequences `\g_@@_notes_in_caption_seq` and `\g_@@_notes_seq` will be merged (in that order) and the notes will be composed.

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

```
515 \newcounter { tabularnote }
```

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

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

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

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

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

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

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

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

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

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

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

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

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

```
531 \cs_set:Npn \thetabularnote { { \@@_notes_style:n { tabularnote } } }
```

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

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

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

```

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

```

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

```

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

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

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

```

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

```

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

```

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

```

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

```

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

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

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

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

```

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

```

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

```

625     \hbox_set:Nn \l_tmpa_box
626     {

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

662         \bool_gset_true:N \g_@@_caption_finished_bool

```



```

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

```

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

```

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

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

```

6 Command for creation of rectangle nodes

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

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

```

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

```

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

```

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

```

```

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

```

7 The options

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

```

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

```

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

```

734 \bool_new:N \l_@@_caption_above_bool

```

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

```

735 \bool_new:N \l_@@_standard_cline_bool

```

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

```

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

```

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

```

738 \bool_new:N \l_@@_xdots_h_labels_bool

```

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

```

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

```

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

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

```

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

```

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

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

```

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

```

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

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

```

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

```

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

```

755 \bool_new:N \l_@@_light_syntax_bool
756 \bool_new:N \l_@@_light_syntax_expanded_bool

```

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

```

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

```

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

```

759 \bool_new:N \l_@@_amp_in_blocks_bool

```

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

```

760 \bool_new:N \l_@@_exterior_arraycolsep_bool

```

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

```

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

```

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

```

763 \clist_new:N \l_@@_corners_clist

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

780 \bool_new:N \l_@@_delimiters_max_width_bool

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

```

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

```

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

```

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

```

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

```

```

826     unknown .code:n =
827       \@@_unknown_key:nn { nicematrix / xdots } { 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 }

```

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

```

837 \keys_define:nn { nicematrix / Global }
838 {
839   caption-above .code:n = \@@_error_or_warning:n { caption-above~in~env } ,
840   show-cell-names .code:n = \@@_error_or_warning:n { show-cell-names } ,
841   color-inside .code:n = \@@_fatal:n { key~color~inside } ,
842   colortbl-like .code:n = \@@_fatal:n { key~color~inside } ,
843   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
844   ampersand-in-blocks .default:n = true ,
845   &-in-blocks .meta:n = ampersand-in-blocks ,
846   no-cell-nodes .code:n =
847     \bool_set_true:N \l_@@_no_cell_nodes_bool
848     \cs_set_protected:Npn \@@_node_cell:
849       { \set@color \box_use_drop:N \l_@@_cell_box } ,
850   no-cell-nodes .value_forbidden:n = true ,
851   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
852   rounded-corners .default:n = 4 pt ,
853   custom-line .code:n = \@@_custom_line:n { #1 } ,
854   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
855   rules .value_required:n = true ,
856   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
857   standard-cline .default:n = true ,
858   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
859   cell-space-top-limit .value_required:n = true ,
860   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
861   cell-space-bottom-limit .value_required:n = true ,
862   cell-space-limits .meta:n =
863     {
864       cell-space-top-limit = #1 ,
865       cell-space-bottom-limit = #1 ,
866     } ,
867   cell-space-limits .value_required:n = true ,
868   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
869   light-syntax .code:n =
870     \bool_set_true:N \l_@@_light_syntax_bool
871     \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
872   light-syntax .value_forbidden:n = true ,
873   light-syntax-expanded .code:n =
874     \bool_set_true:N \l_@@_light_syntax_bool
875     \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
876   light-syntax-expanded .value_forbidden:n = true ,
877   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
878   end-of-row .value_required:n = true ,
879   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
880   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
881   last-row .int_set:N = \l_@@_last_row_int ,
882   last-row .default:n = -1 ,
883   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,

```

```

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

```

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

```

901   { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
902 }
903 { \@@_error:n { One-letter-allowed } }
904 } ,
905 vl_lines-in-sub-matrix .value_required:n = true ,
906 hv_lines .code:n =
907 {
908   \bool_set_true:N \l_@@_hv_lines_bool
909   \tl_set_eq:NN \l_@@_vl_lines_clist \c_@@_all_tl
910   \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
911 } ,
912 hv_lines .value_forbidden:n = true ,
913 hv_lines-except-borders .code:n =
914 {
915   \tl_set_eq:NN \l_@@_vl_lines_clist \c_@@_all_tl
916   \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
917   \bool_set_true:N \l_@@_hv_lines_bool
918   \bool_set_true:N \l_@@_except_borders_bool
919 } ,
920 hv_lines-except-borders .value_forbidden:n = true ,
921 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.

```

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

```

```

942     respect-arraystretch .value_forbidden:n = true ,
943     pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
944     pgf-node-code .value_required:n = true
945 }

```

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

```

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

```

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

```

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

```

970     \legacy_if:nF { measuring@ }
971     {
972         \str_set:Nn \l_@@_name_str { #1 }
973         \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
974         { \@@_err_duplicate_names:n { #1 } }
975         { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
976     } ,
977     name .value_required:n = true ,
978     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
979     code-after .value_required:n = true ,
980 }
981 \cs_set:Npn \@@_err_duplicate_names:n #1
982 { \@@_error:nn { Duplicate~name } { #1 } }
983 \keys_define:nn { nicematrix / notes }
984 {
985     para .bool_set:N = \l_@@_notes_para_bool ,
986     para .default:n = true ,
987     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
988     code-before .value_required:n = true ,
989     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
990     code-after .value_required:n = true ,
991     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,

```



```

992 bottomrule .default:n = true ,
993 style .cs_set:Np = \@@_notes_style:n #1 ,
994 style .value_required:n = true ,
995 label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
996 label-in-tabular .value_required:n = true ,
997 label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
998 label-in-list .value_required:n = true ,
999 enumitem-keys .code:n =
1000 {
1001   \hook_gput_code:nnn { begindocument } { . }
1002   {
1003     \IfPackageLoadedT { enumitem }
1004     { \setlist* [ tabularnotes ] { #1 } }
1005   }
1006 } ,
1007 enumitem-keys .value_required:n = true ,
1008 enumitem-keys-para .code:n =
1009 {
1010   \hook_gput_code:nnn { begindocument } { . }
1011   {
1012     \IfPackageLoadedT { enumitem }
1013     { \setlist* [ tabularnotes* ] { #1 } }
1014   }
1015 } ,
1016 enumitem-keys-para .value_required:n = true ,
1017 detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1018 detect-duplicates .default:n = true ,
1019 unknown .code:n =
1020   \@@_unknown_key:nn { nicematrix / notes } { 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 =
1106     \@@_unknown_key:nn
1107       { nicematrix / Global , nicematrix / NiceMatrixOptions }
1108       { Unknown-key-for-NiceMatrixOptions }
1109 }

```

`\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.

```

1110 \NewDocumentCommand \NiceMatrixOptions { m }
1111 { \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.

```

1112 \keys_define:nn { nicematrix / NiceMatrix }
1113 {
1114   last-col .code:n = \tl_if_empty:nTF { #1 }
1115     {
1116       \bool_set_true:N \l_@@_last_col_without_value_bool
1117       \int_set:Nn \l_@@_last_col_int { -1 }
1118     }
1119     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1120   columns-type .tl_set:N = \l_@@_columns_type_tl ,
1121   columns-type .value_required:n = true ,
1122   l .meta:n = { columns-type = l } ,
1123   r .meta:n = { columns-type = r } ,
1124   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1125   delimiters / color .value_required:n = true ,
1126   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1127   delimiters / max-width .default:n = true ,
1128   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1129   delimiters .value_required:n = true ,
1130   small .bool_set:N = \l_@@_small_bool ,
1131   small .value_forbidden:n = true ,
1132   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1133 }

```

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

```

1134 \keys_define:nn { nicematrix / NiceArray }
1135 {

```

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.

```

1136   small .bool_set:N = \l_@@_small_bool ,
1137   small .value_forbidden:n = true ,
1138   last-col .code:n = \tl_if_empty:nF { #1 }
1139     { \@@_error:n { last-col-non-empty-for-NiceArray } }

```

```

1140         \int_zero:N \l_@@_last_col_int ,
1141     r .code:n = \@@_error:n { r-or~l-with~preamble } ,
1142     l .code:n = \@@_error:n { r-or~l-with~preamble } ,
1143     unknown .code:n =
1144         \@@_unknown_key:nn
1145         { nicematrix / NiceArray , nicematrix / Global , nicematrix / environments}
1146         { Unknown-key~for~NiceArray }
1147 }
1148 \keys_define:nn { nicematrix / pNiceArray }
1149 {
1150     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1151     last-col .code:n = \tl_if_empty:nF { #1 }
1152         { \@@_error:n { last-col-non-empty-for~NiceArray } }
1153         \int_zero:N \l_@@_last_col_int ,
1154     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1155     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1156     delimiters / color .value_required:n = true ,
1157     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1158     delimiters / max-width .default:n = true ,
1159     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1160     delimiters .value_required:n = true ,
1161     small .bool_set:N = \l_@@_small_bool ,
1162     small .value_forbidden:n = true ,
1163     r .code:n = \@@_error:n { r-or~l-with~preamble } ,
1164     l .code:n = \@@_error:n { r-or~l-with~preamble } ,
1165     unknown .code:n =
1166         \@@_unknown_key:nn
1167         { nicematrix / pNiceArray , nicematrix / Global , nicematrix / environments }
1168         { Unknown-key~for~NiceMatrix }
1169 }

```

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

```

1170 \keys_define:nn { nicematrix / NiceTabular }
1171 {

```

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.

```

1172     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1173         \bool_set_true:N \l_@@_width_used_bool ,
1174     width .value_required:n = true ,
1175     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1176     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1177     tabularnote .value_required:n = true ,
1178     caption .tl_set:N = \l_@@_caption_tl ,
1179     caption .value_required:n = true ,
1180     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1181     short-caption .value_required:n = true ,
1182     label .tl_set:N = \l_@@_label_tl ,
1183     label .value_required:n = true ,
1184     last-col .code:n = \tl_if_empty:nF { #1 }
1185         { \@@_error:n { last-col-non-empty-for~NiceArray } }
1186         \int_zero:N \l_@@_last_col_int ,
1187     r .code:n = \@@_error:n { r-or~l-with~preamble } ,
1188     l .code:n = \@@_error:n { r-or~l-with~preamble } ,
1189     unknown .code:n =
1190         \@@_unknown_key:nn
1191         { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
1192         { Unknown-key~for~NiceTabular }
1193 }

```

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
```

```
1194 \keys_define:nn { nicematrix / CodeAfter }
1195 {
1196   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1197   delimiters / color .value_required:n = true ,
1198   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1199   rules .value_required:n = true ,
1200   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1201   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1202   sub-matrix .value_required:n = true ,
1203   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1204 }
```

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}`).

```
1205 \cs_new_protected:Npn \@@_cell_begin:
1206 {
```

`\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.

```
1207   \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).

```
1208   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1209   \cs_set_eq:NN \Hline \@@_Hline_in_cell:
```

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

```
1210   \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.

```
1211   \int_compare:nNnT { \c@jCol } = { \c_one_int }
1212   {
1213     \int_compare:nNnT { \l_@@_first_col_int } = { \c_one_int }
1214     { \@@_begin_of_row: }
1215   }
```

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:`.

```
1216   \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1217   \@@_tuning_not_tabular_begin:
1218   \@@_tuning_first_row:
1219   \@@_tuning_last_row:
1220   \g_@@_row_style_tl
1221 }
```

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.

```
1222 \cs_new_protected:Npn \@@_tuning_first_row:
1223 {
1224   \if_int_compare:w \c@iRow = \c_zero_int
1225     \if_int_compare:w \c@jCol > \c_zero_int
1226       \l_@@_code_for_first_row_tl
1227       \xglobal \colorlet { nicematrix-first-row } { . }
1228     \fi:
1229   \fi:
1230 }
```

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

```
\cs_new_protected:Npn \@@_tuning_last_row:
{
  \int_compare: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.

```
1231 \cs_new_protected:Npn \@@_tuning_last_row:
1232 {
1233   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1234     \l_@@_code_for_last_row_tl
1235     \xglobal \colorlet { nicematrix-last-row } { . }
1236   \fi:
1237 }
```

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

```
1238 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```
1239 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1240 {
1241   \m@th
1242   $ % $
```

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

```
1243 \@@_tuning_key_small:
1244 }
1245 \cs_set:Nn \@@_tuning_not_tabular_end: { $ } % $
```

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

```

1246 \cs_new_protected:Npn \@@_begin_of_row:
1247 {
1248   \int_gincr:N \c@iRow
1249   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1250   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1251   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1252   \pgfpicture
1253   \pgfrememberpicturepositiononpagetrue
1254   \pgfcoordinate
1255   { \@@_env: - row - \int_use:N \c@iRow - base }
1256   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1257   \str_if_empty:NF \l_@@_name_str
1258   {
1259     \pgfnodealias
1260     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1261     { \@@_env: - row - \int_use:N \c@iRow - base }
1262   }
1263   \endpgfpicture
1264 }

```

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.

```

1265 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1266 {
1267   \int_if_zero:nTF { \c@iRow }
1268   {
1269     \dim_compare:nNnT
1270     { \box_dp:N \l_@@_cell_box } > { \g_@@_dp_row_zero_dim }
1271     { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1272     \dim_compare:nNnT
1273     { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_zero_dim }
1274     { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1275   }
1276   {
1277     \int_compare:nNnT { \c@iRow } = { \c_one_int }
1278     {
1279       \dim_compare:nNnT
1280       { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_one_dim }
1281       { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1282     }
1283   }
1284 }
1285 \cs_new_protected:Npn \@@_rotate_cell_box:
1286 {
1287   \box_rotate:Nn \l_@@_cell_box { 90 }
1288   \bool_if:NTF \g_@@_rotate_c_bool
1289   {
1290     \hbox_set:Nn \l_@@_cell_box
1291     {
1292       \m@th
1293       $ % $
1294       \vcenter { \box_use:N \l_@@_cell_box }
1295       $ % $
1296     }
1297   }
1298 }

```

```

1299     \int_compare:nNt { \c@iRow } = { \l_@@_last_row_int }
1300     {
1301         \vbox_set_top:Nn \l_@@_cell_box
1302         {
1303             \vbox_to_zero:n { }
1304             \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1305             \box_use:N \l_@@_cell_box
1306         }
1307     }
1308 }
1309 \bool_gset_false:N \g_@@_rotate_bool
1310 \bool_gset_false:N \g_@@_rotate_c_bool
1311 }
1312 \cs_new_protected:Npn \@@_adjust_size_box:
1313 {
1314     \dim_compare:nNt { \g_@@_blocks_wd_dim } > { \c_zero_dim }
1315     {
1316         \box_set_wd:Nn \l_@@_cell_box
1317         { \dim_max:nn { \box_wd:N \l_@@_cell_box } { \g_@@_blocks_wd_dim } }
1318         \dim_gzero:N \g_@@_blocks_wd_dim
1319     }
1320     \dim_compare:nNt { \g_@@_blocks_dp_dim } > { \c_zero_dim }
1321     {
1322         \box_set_dp:Nn \l_@@_cell_box
1323         { \dim_max:nn { \box_dp:N \l_@@_cell_box } { \g_@@_blocks_dp_dim } }
1324         \dim_gzero:N \g_@@_blocks_dp_dim
1325     }
1326     \dim_compare:nNt { \g_@@_blocks_ht_dim } > { \c_zero_dim }
1327     {
1328         \box_set_ht:Nn \l_@@_cell_box
1329         { \dim_max:nn { \box_ht:N \l_@@_cell_box } { \g_@@_blocks_ht_dim } }
1330         \dim_gzero:N \g_@@_blocks_ht_dim
1331     }
1332 }
1333 \cs_new_protected:Npn \@@_cell_end:
1334 {

```

The following command is nullified in the tabulars.

```

1335     \@@_tuning_not_tabular_end:
1336     \hbox_set_end:
1337     \@@_cell_end_i:
1338 }
1339 \cs_new_protected:Npn \@@_cell_end_i:
1340 {

```

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.

```

1341     \g_@@_cell_after_hook_tl
1342     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
1343     \@@_adjust_size_box:
1344     \box_set_ht:Nn \l_@@_cell_box
1345     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1346     \box_set_dp:Nn \l_@@_cell_box
1347     { \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”).

```

1348     \@@_update_max_cell_width:

```

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

```

1349     \@@_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.

```

1350 \bool_if:NTF \g_@@_empty_cell_bool
1351 { \box_use_drop:N \l_@@_cell_box }
1352 {
1353   \bool_if:NTF \g_@@_not_empty_cell_bool
1354   { \@@_print_node_cell: }
1355   {
1356     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
1357     { \@@_print_node_cell: }
1358     { \box_use_drop:N \l_@@_cell_box }
1359   }
1360 }
1361 \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
1362 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1363 \bool_gset_false:N \g_@@_empty_cell_bool
1364 \bool_gset_false:N \g_@@_not_empty_cell_bool
1365 }

```

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

```

1366 \cs_new_protected:Npn \@@_update_max_cell_width:
1367 {
1368   \dim_gset:Nn \g_@@_max_cell_width_dim
1369   { \dim_max:nn { \g_@@_max_cell_width_dim } { \box_wd:N \l_@@_cell_box } }
1370 }

```

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`).

```

1371 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1372 {
1373   \@@_math_toggle:
1374   \hbox_set_end:
1375   \bool_if:NF \g_@@_rotate_bool
1376   {
1377     \hbox_set:Nn \l_@@_cell_box
1378     {
1379       \makebox [ \l_@@_col_width_dim ] [ s ]
1380       { \hbox_unpack_drop:N \l_@@_cell_box }
1381     }
1382   }
1383   \@@_cell_end_i:
1384 }

```

```

1385 \pgfset
1386 {
1387     nicematrix / cell-node /.style =
1388     {
1389         inner-sep = \c_zero_dim ,
1390         minimum-width = \c_zero_dim
1391     }
1392 }

```

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.

```

1393 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1394 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1395 {
1396     \use:c
1397     {
1398         __siunitx_table_align_
1399         \bool_if:NTF \l__siunitx_table_text_bool
1400             { \l__siunitx_table_align_text_tl }
1401             { \l__siunitx_table_align_number_tl }
1402         :n
1403     }
1404     { #1 }
1405 }

```

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!

```

1406 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1407 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1408 {
1409     \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1410     \hbox:n
1411     {
1412         \pgfsys@markposition
1413         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1414     }
1415     #1
1416     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1417     \hbox:n
1418     {
1419         \pgfsys@markposition
1420         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1421     }
1422 }

1423 \cs_new_protected:Npn \@@_print_node_cell:
1424 {
1425     \socket_use:nn { nicematrix / siunitx-wrap }
1426     { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1427 }

```

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

```

1428 \cs_new_protected:Npn \@@_node_cell:
1429 {
1430     \pgfpicture
1431     \pgfsetbaseline \c_zero_dim

```

```

1432 \pgfrememberpicturepositiononpagetrue
1433 \pgfset { nicematrix / cell-node }
1434 \pgfnode
1435 { rectangle }
1436 { base }
1437 {

```

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).

```

1438 \sys_if_engine_xetex:T { \set@color }
1439 \box_use:N \l_@@_cell_box
1440 }
1441 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1442 { \l_@@_pgf_node_code_tl }
1443 \str_if_empty:NF \l_@@_name_str
1444 {
1445 \pgfnodealias
1446 { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1447 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1448 }
1449 \endpgfpicture
1450 }

```

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).

```

1451 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1452 {
1453 \bool_if:nTF { #1 } { \tl_gput_left:ce } { \tl_gput_right:ce }
1454 { g_@@_ #2 _ lines _ tl }
1455 {
1456 \use:c { @@ _ draw _ #2 : nnn }
1457 { \int_use:N \c@iRow }
1458 { \int_use:N \c@jCol }
1459 { \exp_not:n { #3 } }
1460 }
1461 }

1462 \cs_new_protected:Npn \@@_array:n
1463 {
1464 \dim_set:Nn \col@sep
1465 { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1466 \dim_compare:nNnTF { \l_@@_tabular_width_dim } = { \c_zero_dim }
1467 { \def \@halignto { } }
1468 { \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.

```
1469   \@tabarray
\l_@@_baseline_tl may have the value t, c or b. However, if the value is b, we compose
the \array (of array) with the option t and the right translation will be done further. Re-
mark that \str_if_eq:eeTF is fully expandable and we need something fully expandable here.
\str_if_eq:ee(TF) is faster than \str_if_eq:nn(TF).
```

```
1470   [ \str_if_eq:eeTF { \l_@@_baseline_tl } { c } { c } { t } ]
1471   }
1472   \cs_generate_variant:Nn \@@_array:n { o }
```

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

```
1473   \bool_if:NTF \c_@@_revtex_bool
1474   { \cs_set_eq:NN \@@_old_ialign: \ialign }
```

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

```
1475   { \cs_set_eq:cN { @@_old_ar@ialign: } \ar@ialign }
```

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

```
1476   \cs_new_protected:Npn \@@_create_row_node:
1477   {
1478     \int_compare:nNt { \c@iRow } > { \g_@@_last_row_node_int }
1479     {
1480       \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1481       \@@_create_row_node_i:
1482     }
1483   }
1484   \cs_new_protected:Npn \@@_create_row_node_i:
1485   {
```

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

```
1486     \hbox
1487     {
1488       \bool_if:NT \l_@@_code_before_bool
1489       {
1490         \vtop
1491         {
1492           \skip_vertical:N 0.5\arrayrulewidth
1493           \pgfsys@markposition
1494           { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1495           \skip_vertical:N -0.5\arrayrulewidth
1496         }
1497       }
1498       \pgfpicture
1499       \pgfrememberpicturerepositiononpagetrue
1500       \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1501       { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1502       \str_if_empty:NF \l_@@_name_str
1503       {
1504         \pgfnodealias
1505         { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1506         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1507       }
1508       \endpgfpicture
1509     }
1510   }
```

```
1511   \cs_new_protected:Npn \@@_in_everycr:
1512   {
1513     \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
```

```

1514 \tbl_update_cell_data_for_next_row:
1515 \int_gzero:N \c@jCol
1516 \bool_gset_false:N \g_@@_after_col_zero_bool
1517 \bool_if:NF \g_@@_row_of_col_done_bool
1518 {
1519 \@@_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).

```

1520 \clist_if_empty:NF \l_@@_hlines_clist
1521 {
1522 \str_if_eq:eeF { \l_@@_hlines_clist } { all }
1523 {
1524 \clist_if_in:NtT
1525 \l_@@_hlines_clist
1526 { \int_eval:n { \c@iRow + 1 } }
1527 }
1528 {

```

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.

```

1529 \int_compare:nNnT { \c@iRow } > { -1 }
1530 {
1531 \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
1532 { \hrule height \arrayrulewidth width \c_zero_dim }
1533 }
1534 }
1535 }
1536 }
1537 }

```

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

```

1538 \cs_set_protected:Npn \@@_renew_dots:
1539 {
1540 \cs_set_eq:NN \ldots \@@_Ldots:
1541 \cs_set_eq:NN \cdots \@@_Cdots:
1542 \cs_set_eq:NN \vdots \@@_Vdots:
1543 \cs_set_eq:NN \ddots \@@_Ddots:
1544 \cs_set_eq:NN \iddots \@@_Iddots:
1545 \cs_set_eq:NN \dots \@@_Ldots:
1546 \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1547 }

```

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

```

1548 \hook_gput_code:nnn { begindocument } { . }
1549 {
1550 \IfPackageLoadedTF { booktabs }
1551 {
1552 \cs_new_protected:Npn \@@_patch_booktabs:
1553 { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1554 }
1555 { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1556 }

```

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

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`.

```

1557 \cs_new_protected:Npn \@_some_initialization:
1558 {
1559   \@@_everycr:
1560   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1561   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1562   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1563   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1564   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1565   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1566 }

```

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

```

1567 \cs_new_protected:Npn \@_pre_array_after_CodeBefore:
1568 {

```

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.

```

1569   \seq_gset_eq:NN \g_@@_pos_of_blocks_seq \g_@@_future_pos_of_blocks_seq
1570   \seq_gclear:N \g_@@_future_pos_of_blocks_seq

```

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

```

1571   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1572   \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.

```

1573   \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1574   \fp_gzero:N \g_@@_total_X_weight_fp
1575   \bool_gset_false:N \g_@@_V_of_X_bool
1576   \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1577   \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol
1578   \@@_patch_booktabs:
1579   \box_clear_new:N \l_@@_cell_box
1580   \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}`).

```

1581   \bool_if:NT \l_@@_small_bool
1582   {

```

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

```

1583     \def \arraystretch { 0.47 }
1584     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1585     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1586 }

```

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!

```

1587     \bool_if:NT \g_@@_create_cell_nodes_bool
1588     {
1589         \tl_put_right:Nn \@@_begin_of_row:
1590         {
1591             \pgfsys@markposition
1592             { \@@_env: - row - \int_use:N \c@iRow - base }
1593         }
1594         \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1595     }

```

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

```

1596     \bool_if:NF \c_@@_revtex_bool
1597     {
1598         \def \ar@ialign
1599         {
1600             \tbl_init_cell_data_for_table:
1601             \@@_some_initialization:
1602             \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`.

```

1603             \cs_set_eq:Nc \ar@ialign { @@_old_ar@ialign: }
1604             \halign
1605         }
1606     }

```

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).

```

1607     \bool_if:NT \c_@@_revtex_bool
1608     {
1609         \IfPackageLoadedT { colortbl }
1610         { \cs_set_protected:Npn \CT@setup { } }
1611     }

```

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).

```

1612     \cs_set_eq:NN \@@_old_ldots: \ldots
1613     \cs_set_eq:NN \@@_old_cdots: \cdots
1614     \cs_set_eq:NN \@@_old_vdots: \vdots
1615     \cs_set_eq:NN \@@_old_ddots: \ddots
1616     \cs_set_eq:NN \@@_old_iddots: \iddots
1617     \bool_if:NTF \l_@@_standard_cline_bool
1618     { \cs_set_eq:NN \cline \@@_standard_cline: }
1619     { \cs_set_eq:NN \cline \@@_cline: }
1620     \cs_set_eq:NN \Ldots \@@_Ldots:

```

```

1621 \cs_set_eq:NN \Cdots \@@_Cdots:
1622 \cs_set_eq:NN \Vdots \@@_Vdots:
1623 \cs_set_eq:NN \Ddots \@@_Ddots:
1624 \cs_set_eq:NN \Iddots \@@_Iddots:
1625 \cs_set_eq:NN \Hline \@@_Hline:
1626 \cs_set_eq:NN \Hspace \@@_Hspace:
1627 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1628 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1629 \cs_set_eq:NN \Block \@@_Block:
1630 \cs_set_eq:NN \rotate \@@_rotate:
1631 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1632 \cs_set_eq:NN \dotfill \@@_dotfill:
1633 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1634 \cs_if_free:NT \Body { \cs_set_eq:NN \Body \@@_Body: }
1635 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1636 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1637 \cs_set_eq:NN \TopRule \@@_TopRule
1638 \cs_set_eq:NN \MidRule \@@_MidRule
1639 \cs_set_eq:NN \BottomRule \@@_BottomRule
1640 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1641 \cs_set_eq:NN \Hbrace \@@_Hbrace
1642 \cs_set_eq:NN \Vbrace \@@_Vbrace
1643 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1644   { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1645 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1646 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1647 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1648 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1649 \int_compare:nNnT { \l_@@_first_row_int } > { \c_zero_int }
1650   { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1651 \int_compare:nNnT { \l_@@_last_row_int } < { \c_zero_int }
1652   { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1653 \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:`.

```

1654 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1655 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1656   { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1657 \@@_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!).

```

1658 \tl_if_exist:NT \l_@@_note_in_caption_tl
1659   {
1660     \tl_if_empty:NF \l_@@_note_in_caption_tl
1661     {
1662       \int_gset:Nn \g_@@_notes_caption_int { \l_@@_note_in_caption_tl }
1663       \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1664     }
1665   }

```

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).

```

1666 \seq_gclear:N \g_@@_multicolumn_cells_seq
1667 \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).

```
1668 \int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }
```

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

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

```
1669 \int_gzero:N \g_@@_row_total_int
```

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

```
1670 \int_gzero:N \g_@@_col_total_int
1671 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1672 \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.

```
1673 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1674 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1675 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1676 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1677 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1678 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1679 \tl_gclear:N \g_nicematrix_code_before_tl
1680 \tl_gclear:N \g_@@_pre_code_before_tl
```

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

```
1681 \dim_zero_new:N \l_@@_left_delim_dim
1682 \dim_zero_new:N \l_@@_right_delim_dim
1683 \bool_if:NTF \g_@@_delims_bool
1684 {
```

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

```
1685 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1686 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1687 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1688 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1689 }
1690 {
1691 \dim_gset:Nn \l_@@_left_delim_dim
1692 { 2 \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1693 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1694 }
1695 }
```

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

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

```
1696 \cs_new_protected:Npn \@@_pre_array:
1697 {
1698 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1699 \int_gzero_new:N \c@iRow
1700 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1701 \int_gzero_new:N \c@jCol
```

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

```

1702 \int_compare:nNnT \l_@@_last_row_int > 0
1703   { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1704 \int_compare:nNnT \l_@@_last_col_int > 0
1705   { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1706 \bool_if:NT \g_@@_aux_found_bool
1707   {
1708     \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1709     \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1710     \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1711     \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1712   }

```

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

```

1713 \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
1714   {
1715     \bool_set_true:N \l_@@_last_row_without_value_bool
1716     \bool_if:NT \g_@@_aux_found_bool
1717       { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1718   }
1719 \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
1720   {
1721     \bool_if:NT \g_@@_aux_found_bool
1722       { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1723   }

```

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”.

```

1724 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
1725   {
1726     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1727       {
1728         \dim_compare:nNnT { \g_@@_ht_last_row_dim } < { \box_ht:N \l_@@_cell_box }
1729           { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1730         \dim_compare:nNnT { \g_@@_dp_last_row_dim } < { \box_dp:N \l_@@_cell_box }
1731           { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1732       }
1733   }

1734 \seq_gclear:N \g_@@_cols_vlism_seq
1735 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1736 \bool_if:NT \l_@@_code_before_bool { \@@_exec_code_before: }

```

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

```

1737 \@@_pre_array_after_CodeBefore:

```

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

```
1738 \hbox_set:Nw \l_@@_the_array_box
1739 \skip_horizontal:N \l_@@_left_margin_dim
1740 \skip_horizontal:N \l_@@_extra_left_margin_dim
1741 \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).

```
1742 \m@th
1743 $ % $
1744 \bool_if:NTF \l_@@_light_syntax_bool
1745 { \use:c { @@-light-syntax } }
1746 { \use:c { @@-normal-syntax } }
1747 }
```

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

```
1748 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1749 {
1750   \tl_set:Nn \l_tmpa_tl { #1 }
1751   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1752     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1753   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1754   \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...

```
1755 \@@_pre_array:
1756 }
```

9 The `\CodeBefore`

```
1757 \cs_new_protected_nopar:Npn \@@_Body: { \@@_fatal:n { Body~alone } }
```

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

```
1758 \cs_new_protected:Npn \@@_pre_code_before:
1759 {
```

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.

```
1760 \pgfsys@markposition { \@@_env: - position }
1761 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1762 \pgfpicture
1763 \pgf@relevantforpicturesizefalse
```

First, the recreation of the row nodes.

```
1764 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int + 1 }
1765 {
1766   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1767   \pgfcoordinate { \@@_env: - row - ##1 }
1768     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1769 }
```

Now, the recreation of the `col` nodes.

```
1770 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int + 1 }
1771 {
1772   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
```

```

1773     \pgfcoordinate { \@@_env: - col - ##1 }
1774     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1775 }

```

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

```

1776 \@@_create_diag_nodes:

```

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

```

1777 \bool_if:NT \g_@@_create_cell_nodes_bool { \@@_recreate_cell_nodes: }
1778 \endpgfpicture

```

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

```

1779 \@@_create_blocks_nodes:
1780 \IfPackageLoadedT { tikz }
1781 {
1782     \tikzset
1783     {
1784         every-picture / .style =
1785         { overlay , name~prefix = \@@_env: - }
1786     }
1787 }
1788 \cs_set_eq:NN \cellcolor \@@_cellcolor
1789 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1790 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1791 \cs_set_eq:NN \rowcolor \@@_rowcolor
1792 \cs_set_eq:NN \rowcolors \@@_rowcolors
1793 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1794 \cs_set_eq:NN \arraycolor \@@_arraycolor
1795 \cs_set_eq:NN \columncolor \@@_columncolor
1796 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1797 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1798 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1799 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1800 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1801 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1802 }

```

```

1803 \cs_new_protected:Npn \@@_exec_code_before:
1804 {

```

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...

```

1805 \clist_map_inline:Nn \l_@@_corners_cells_clist
1806 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1807 \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`.

```

1808 \@@_add_to_colors_seq:nn { { nocolor } } { }
1809 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1810 \group_begin:

```

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

```

1811 \if_mode_math:
1812     \@@_exec_code_before_i:
1813 \else:
1814     $ % $
1815     \@@_exec_code_before_i:
1816     $ % $
1817 \fi:
1818 \group_end:
1819 }

```

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`).

```

1820 \cs_new_protected:Npn \@@_exec_code_before_i:
1821 {
1822   \int_compare:nNt { \char_value_catcode:n { 60 } } = { 13 }
1823   { \@@_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`.

```

1824   \exp_last_unbraced:No \@@_CodeBefore_keys:
1825   \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.

```

1826   \@@_actually_color:
1827   \l_@@_code_before_tl
1828   \q_stop
1829 }

```

```

1830 \keys_define:nn { nicematrix / CodeBefore }
1831 {
1832   create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1833   create-cell-nodes .default:n = true ,
1834   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1835   sub-matrix .value_required:n = true ,
1836   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1837   delimiters / color .value_required:n = true ,
1838   unknown .code:n = \@@_error:n { Unknown-key-for-CodeBefore }
1839 }

1840 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1841 {
1842   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1843   \@@_CodeBefore:w
1844 }

```

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.

```

1845 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1846 {
1847   \bool_if:NT \g_@@_aux_found_bool
1848   {
1849     \@@_pre_code_before:
1850     \legacy_if:nF { measuring@ } { #1 }
1851   }
1852 }

```

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.

```

1853 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1854 {
1855   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
1856   {
1857     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1858     \pgfcoordinate { \@@_env: - row - ##1 - base }
1859     { \pgfpointdiff \@@_picture_position: \@@_node_position: }

```

```

1860 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
1861 {
1862   \cs_if_exist:cT
1863   { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1864   {
1865     \pgfsys@getposition
1866     { \@@_env: - ##1 - #####1 - NW }
1867     \@@_node_position:
1868     \pgfsys@getposition
1869     { \@@_env: - ##1 - #####1 - SE }
1870     \@@_node_position_i:
1871     \@@_pgf_rect_node:nnn
1872     { \@@_env: - ##1 - #####1 }
1873     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1874     { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1875   }
1876 }
1877 }
1878 \@@_create_extra_nodes:
1879 \@@_create_aliases_last:
1880 }

1881 \cs_new_protected:Npn \@@_create_aliases_last:
1882 {
1883   \int_step_inline:nn { \c@iRow }
1884   {
1885     \pgfnodealias
1886     { \@@_env: - ##1 - last }
1887     { \@@_env: - ##1 - \int_use:N \c@jCol }
1888   }
1889   \int_step_inline:nn { \c@jCol }
1890   {
1891     \pgfnodealias
1892     { \@@_env: - last - ##1 }
1893     { \@@_env: - \int_use:N \c@iRow - ##1 }
1894   }
1895   \pgfnodealias
1896   { \@@_env: - last - last }
1897   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1898 }

1899 \cs_new_protected:Npn \@@_create_blocks_nodes:
1900 {
1901   \pgfpicture
1902   \pgf@relevantforpicturesizefalse
1903   \pgfrememberpicturepositiononpagetrue
1904   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1905   { \@@_create_one_block_node:nnnnn ##1 }
1906   \endpgfpicture
1907 }

```

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

```

1908 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1909 {
1910   \tl_if_empty:nF { #5 }
1911   {
1912     \@@_qpoint:n { col - #2 }
1913     \dim_set_eq:NN \l_tmpa_dim \pgf@x

```

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).

```

1914 \@@_qpoint:n { #1 }
1915 \dim_set_eq:NN \l_tmpb_dim \pgf@y
1916 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1917 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1918 \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1919 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1920 \@@_pgf_rect_node:nnnnn
1921 { \@@_env: - #5 }
1922 { \dim_use:N \l_tmpa_dim }
1923 { \dim_use:N \l_tmpb_dim }
1924 { \dim_use:N \l_@@_tmpc_dim }
1925 { \dim_use:N \l_@@_tmpd_dim }
1926 }
1927 }

1928 \cs_new_protected:Npn \@@_patch_for_revtext:
1929 {
1930 \cs_set_eq:NN \@addamp \@addamp@LaTeX
1931 \cs_set_eq:NN \@array \@array@array
1932 \cs_set_eq:NN \@tabular \@tabular@array
1933 \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1934 \cs_set_eq:NN \array \array@array
1935 \cs_set_eq:NN \endarray \endarray@array
1936 \cs_set:Npn \endtabular { \endarray $\egroup } % $
1937 \cs_set_eq:NN \@mkpream \@mkpream@array
1938 \cs_set_eq:NN \@classx \@classx@array
1939 \cs_set_eq:NN \insert@column \insert@column@array
1940 \cs_set_eq:NN \@arraycr \@arraycr@array
1941 \cs_set_eq:NN \@xarraycr \@xarraycr@array
1942 \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1943 }

```

10 The environment `{NiceArrayWithDelims}`

```

1944 \NewDocumentEnvironment { NiceArrayWithDelims }
1945 { m m O { } m ! O { } t \CodeBefore }
1946 {
1947 \bool_if:NT \c_@@_revtex_bool { \@@_patch_for_revtext: }
1948 \@@_provide_pgfsyspdfmark:
1949 \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.

```

1950 \bgroup

1951 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1952 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1953 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1954 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

1955 \int_gzero:N \g_@@_block_box_int
1956 \dim_gzero:N \g_@@_width_last_col_dim
1957 \dim_gzero:N \g_@@_width_first_col_dim
1958 \bool_gset_false:N \g_@@_row_of_col_done_bool
1959 \str_if_empty:NT \g_@@_name_env_str
1960 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1961 \bool_if:NTF \l_@@_tabular_bool
1962 { \mode_leave_vertical: }
1963 { \@@_test_if_math_mode: }
1964 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1965 \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.

```
1966 \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.

```
1967 \cs_if_exist:NT \tikz@library@external@loaded
1968 {
1969     \tikzexternaldisable
1970     \cs_if_exist:NT \ifstandalone
1971     { \tikzset { external / optimize = false } }
1972 }
```

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

```
1973 \int_gincr:N \g_@@_env_int
1974 \bool_if:NF \l_@@_block_auto_columns_width_bool
1975 { \dim_gzero_new:N \g_@@_max_cell_width_dim }
```

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

```
1976 \seq_gclear:N \g_@@_blocks_seq
1977 \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`.

```
1978 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1979 \seq_gclear:N \g_@@_pos_of_xdots_seq
1980 \tl_gclear_new:N \g_@@_code_before_tl
1981 \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.

```
1982 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1983 {
1984     \bool_gset_true:N \g_@@_aux_found_bool
1985     \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1986 }
1987 { \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.

```
1988 \tl_gclear:N \g_@@_aux_tl
1989 \tl_if_empty:NF \g_@@_code_before_tl
1990 {
1991     \bool_set_true:N \l_@@_code_before_bool
1992     \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1993 }
1994 \tl_if_empty:NF \g_@@_pre_code_before_tl
1995 { \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`.

```
1996 \bool_if:NTF \g_@@_delims_bool
1997 { \keys_set:nn { nicematrix / pNiceArray } }
1998 { \keys_set:nn { nicematrix / NiceArray } }
1999 { #3 , #5 }
```

⁸e.g. `\color[rgb]{0.5,0.5,0}`


```
2000 \@@_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:.

```
2001 \bool_if:nTF { #6 } { \@@_CodeBefore_Body:w } { \@@_pre_array: }
2002 }
```

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

```
2003 {
2004   \bool_if:nTF \l_@@_light_syntax_bool
2005     { \use:c { end @@-light-syntax } }
2006     { \use:c { end @@-normal-syntax } }
2007   $ % $
2008   \skip_horizontal:N \l_@@_right_margin_dim
2009   \skip_horizontal:N \l_@@_extra_right_margin_dim
2010   \hbox_set_end:
2011   \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.

```
2012 \bool_if:NT \l_@@_width_used_bool
2013 {
2014   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2015     { \@@_error_or_warning:n { width-without-X-columns } }
2016 }
```

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 $\text{l_@@_X_columns_dim}$ multiplied by x .

```
2017 \fp_compare:nNnT { \g_@@_total_X_weight_fp } > { \c_zero_fp }
2018   { \@@_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).

```
2019 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
2020 {
2021   \bool_if:NF \l_@@_last_row_without_value_bool
2022   {
2023     \int_compare:nNnF { \l_@@_last_row_int } = { \c@iRow }
2024     {
2025       \@@_error:n { Wrong-last-row }
2026       \int_set_eq:NN \l_@@_last_row_int \c@iRow
2027     }
2028   }
2029 }
```

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

```
2030 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2031 \bool_if:NTF \g_@@_last_col_found_bool
2032   { \int_gdecr:N \c@jCol }
2033   {
2034     \int_compare:nNnT { \l_@@_last_col_int } > { -1 }
2035     { \@@_error:n { last-col-not-used } }
```

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

2036 }

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

2037 \int_gset:nn \g_@@_row_total_int \c@iRow
2038 \int_compare:nNt { \l_@@_last_row_int } > { -1 }
2039 { \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).

2040 \int_if_zero:nT { \l_@@_first_col_int }
2041 { \skip_horizontal:N \g_@@_width_first_col_dim }

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

2042 \bool_if:nTF { ! \g_@@_delims_bool }
2043 {
2044 \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
2045 { \@@_use_arraybox_with_notes_c: }
2046 {
2047 \str_if_eq:eeTF { \l_@@_baseline_tl } { b }
2048 { \@@_use_arraybox_with_notes_b: }
2049 { \@@_use_arraybox_with_notes: }
2050 }
2051 }

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).

2052 {
2053 \int_if_zero:nTF { \l_@@_first_row_int }
2054 {
2055 \dim_set_eq:nn \l_tmpa_dim \g_@@_dp_row_zero_dim
2056 \dim_add:nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2057 }
2058 { \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

2059 \int_compare:nNtTF { \l_@@_last_row_int } > { -2 }
2060 {
2061 \dim_set_eq:nn \l_tmpb_dim \g_@@_ht_last_row_dim
2062 \dim_add:nn \l_tmpb_dim \g_@@_dp_last_row_dim
2063 }
2064 { \dim_zero:N \l_tmpb_dim }
2065 \hbox_set:Nn \l_tmpa_box
2066 {
2067 \m@th
2068 \$ % \$
2069 \@@_color:o \l_@@_delimiters_color_tl
2070 \exp_after:wN \left \g_@@_left_delim_tl
2071 \vcenter
2072 {

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.

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

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).

```

2081         { \skip_horizontal:n { - \tabcolsep } }
2082         { \skip_horizontal:n { - \arraycolsep } }
2083     }

```

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

```

2084         \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2085     }
2086     \exp_after:wN \right \g_@@_right_delim_tl
2087     $ % $
2088 }

```

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.

```

2089     \bool_if:NTF \l_@@_delimiters_max_width_bool
2090     {
2091         \@@_put_box_in_flow_bis:nn
2092         { \g_@@_left_delim_tl }
2093         { \g_@@_right_delim_tl }
2094     }
2095     \@@_put_box_in_flow:
2096 }

```

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).

```

2097     \bool_if:NT \g_@@_last_col_found_bool
2098     { \skip_horizontal:N \g_@@_width_last_col_dim }
2099     \bool_if:NT \l_@@_preamble_bool
2100     {
2101         \int_compare:nNnT { \c@jCol } < { \g_@@_static_num_of_col_int }
2102         { \@@_err_columns_not_used: }
2103     }
2104     \@@_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.

```

2105     \egroup

```

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

```

2106     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2107     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2108     \iow_now:Ne \@mainaux
2109     {
2110         \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2111         \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2112         { \exp_not:o \g_@@_aux_tl }
2113     }
2114     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2115     \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2116 }

```

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

```

2117 \cs_new_protected:Npn \@@_err_columns_not_used:
2118 {
2119     \@@_warning:n { columns~not~used }
2120     \cs_gset:Npn \@@_err_columns_not_used: { }
2121 }

```

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 .

```

2122 \cs_new_protected:Npn \@@_compute_width_X:
2123 {
2124   \tl_gput_right:Nx \g_@@_aux_tl
2125   {
2126     \bool_set_true:N \l_@@_X_columns_aux_bool
2127     \dim_set:Nn \l_@@_X_columns_dim
2128     {

```

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`

```

2129       \bool_lazy_and:nnTF
2130       { \g_@@_V_of_X_bool }
2131       { \l_@@_X_columns_aux_bool }
2132       { \dim_use:N \l_@@_X_columns_dim }
2133       {
2134         \dim_compare:nNnTF
2135         {
2136           \dim_abs:n
2137           { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2138         }
2139         <
2140         { 0.001 pt }
2141         { \dim_use:N \l_@@_X_columns_dim }
2142         {
2143           \dim_eval:n
2144           {
2145             \l_@@_X_columns_dim
2146             +
2147             \fp_to_dim:n
2148             {
2149               (
2150                 \dim_eval:n
2151                 { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2152               )
2153               / \fp_use:N \g_@@_total_X_weight_fp
2154             }
2155           }
2156         }
2157       }
2158     }
2159   }
2160 }

```

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`.

```

2161 \cs_new_protected:Npn \@@_transform_preamble:
2162 {
2163   \@@_transform_preamble_i:
2164   \@@_transform_preamble_ii:
2165 }
2166 \cs_new_protected:Npn \@@_transform_preamble_i:
2167 {
2168   \int_gzero:N \c@jCol

```

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

```
2169 \seq_gclear:N \g_@@_cols_vlism_seq
```

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

```
2170 \bool_gset_false:N \g_tmpb_bool
```

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

```
2171 \tl_gclear_new:N \g_@@_pre_cell_tl
```

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

```
2172 \int_zero:N \l_tmpa_int
2173 \tl_gclear:N \g_@@_array_preamble_tl
2174 \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2175 {
2176   \tl_gset:Nn \g_@@_array_preamble_tl
2177     { ! { \skip_horizontal:N \arrayrulewidth } }
2178 }
2179 {
2180   \clist_if_in:NnT \l_@@_vlines_clist 1
2181   {
2182     \tl_gset:Nn \g_@@_array_preamble_tl
2183       { ! { \skip_horizontal:N \arrayrulewidth } }
2184   }
2185 }
```

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

```
2186 \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2187 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2188 \@@_replace_columncolor:
2189 }

2190 \cs_new_protected:Npn \@@_transform_preamble_ii:
2191 {
```

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

```
2192 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2193 {
2194   \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2195   { \bool_gset_true:N \g_@@_delims_bool }
2196 }
2197 { \bool_gset_true:N \g_@@_delims_bool }
```

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

```
2198 \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).

```
2199 \int_if_zero:nTF { \l_@@_first_col_int }
2200 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2201 {
2202   \bool_if:NF \g_@@_delims_bool
2203   {
2204     \bool_if:NF \l_@@_tabular_bool
2205     {
2206       \clist_if_empty:NT \l_@@_vlines_clist
2207       {
2208         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2209         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
```

```

2210     }
2211   }
2212 }
2213 }
2214 \int_compare:nNnTF { \l_@@_last_col_int } > { -1 }
2215 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2216 {
2217   \bool_if:NF \g_@@_delims_bool
2218   {
2219     \bool_if:NF \l_@@_tabular_bool
2220     {
2221       \clist_if_empty:NT \l_@@_vlines_clist
2222       {
2223         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2224         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2225       }
2226     }
2227   }
2228 }

```

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

```

2229   \tag_if_active:F
2230   {

```

Moreover, when {NiceTabular*} is used, the mechanism can't be used for technical reasons. We test that situation with \l_@@_tabular_width_dim.

```

2231   \dim_compare:nNnT { \l_@@_tabular_width_dim } = { \c_zero_dim }
2232   {
2233     \tl_gput_right:Nn \g_@@_array_preamble_tl
2234     { > { \@@_err_too_many_cols: } 1 }
2235   }
2236 }
2237 }

```

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

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

```

2238 \cs_new_protected:Npn \@@_rec_preamble:n #1
2239 {

```

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

```

2240   \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2241   { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2242   {

```

Now, the columns defined by \newcolumntype of array.

```

2243   \cs_if_exist:cTF { NC @ find @ #1 }
2244   {
2245     \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2246     \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2247   }
2248   {

```

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.

```

2249         \str_if_eq:nnTF { #1 } { S }
2250         { \@@_fatal:n { unknown~column~type~S } }
2251         { \@@_fatal:nn { unknown~column~type } { #1 } }
2252     }
2253 }
2254 }

```

For c, l and r

```

2255 \cs_new_protected:Npn \@@_c: #1
2256 {
2257     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2258     \tl_gclear:N \g_@@_pre_cell_tl
2259     \tl_gput_right:Nn \g_@@_array_preamble_tl
2260     { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2261     \int_gincr:N \c@jCol
2262     \@@_rec_preamble_after_col:n
2263 }

2264 \cs_new_protected:Npn \@@_l: #1
2265 {
2266     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2267     \tl_gclear:N \g_@@_pre_cell_tl
2268     \tl_gput_right:Nn \g_@@_array_preamble_tl
2269     {
2270         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2271         l
2272         < \@@_cell_end:
2273     }
2274     \int_gincr:N \c@jCol
2275     \@@_rec_preamble_after_col:n
2276 }

2277 \cs_new_protected:Npn \@@_r: #1
2278 {
2279     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2280     \tl_gclear:N \g_@@_pre_cell_tl
2281     \tl_gput_right:Nn \g_@@_array_preamble_tl
2282     {
2283         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2284         r
2285         < \@@_cell_end:
2286     }
2287     \int_gincr:N \c@jCol
2288     \@@_rec_preamble_after_col:n
2289 }

```

For ! and @

```

2290 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2291 {
2292     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2293     \@@_rec_preamble:n
2294 }
2295 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2296 \cs_new_protected:cpn { @@ _ | : } #1
2297 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2298     \int_incr:N \l_tmpa_int
2299     \@@_make_preamble_i:n
2300 }

```

```

2301 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2302 {

```

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

```

2303 \str_if_eq:nnTF { #1 } { | }
2304 { \use:c { @@ _ | : } | }
2305 { \@@_make_preamble_i_ii:nn { } #1 }
2306 }

```

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

```

2307 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2308 {
2309 \str_if_eq:nnTF { #2 } { [ ]
2310 { \@@_make_preamble_i_ii:nw { #1 } [ ]
2311 { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2312 }
2313 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2314 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2315 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2316 {
2317 \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2318 \tl_gput_right:Ne \g_@@_array_preamble_tl
2319 {

```

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

```

2320 \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2321 }
2322 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2323 {
2324 \@@_vline:n
2325 {
2326 position = \int_eval:n { \c_jCol + 1 } ,
2327 multiplicity = \int_use:N \l_tmpa_int ,
2328 total-width = \dim_use:N \l_@@_rule_width_dim ,
2329 #2
2330 }
2331 }

```

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.

```

2331 }
2332 \int_zero:N \l_tmpa_int
2333 \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2334 \@@_rec_preamble:n #1
2335 }

```

```

2336 \cs_new_protected:cpn { @@ _ > : } #1 #2
2337 {
2338 \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2339 \@@_rec_preamble:n
2340 }
2341 \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.

```

2342 \keys_define:nn { nicematrix / p-column }
2343 {
2344 r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2345 r .value_forbidden:n = true ,
2346 c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2347 c .value_forbidden:n = true ,
2348 l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,

```



```

2349 l .value_forbidden:n = true ,
2350 S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2351 S .value_forbidden:n = true ,
2352 p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2353 p .value_forbidden:n = true ,
2354 t .meta:n = p ,
2355 m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2356 m .value_forbidden:n = true ,
2357 b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2358 b .value_forbidden:n = true
2359 }

```

For p but also b and m.

```

2360 \cs_new_protected:Npn \@@_p: #1
2361 {
2362   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

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

```

2375 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2376 {

```

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).

```

2377   \str_set:Nn \l_@@_hpos_col_str { j }
2378   \@@_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`.

```

2379   \setlength { \l_tmpa_dim } { #2 }
2380   \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2381 }
2382 \cs_new_protected:Npn \@@_keys_p_column:n #1
2383 { \keys_set_known:nnN { nicematrix / p-column } { #1 } \l_tmpa_tl }

```

The first argument is the width of the column. The second is the type of environment: `minipage` or `varwidth`. The third is some code added at the beginning of the cell.

```

2384 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2385 {

```

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

```

2386   \use:e
2387   {
2388     \@@_make_preamble_ii_vi:nnnnnnnn
2389     { \str_if_eq:eeTF { \l_@@_vpos_col_str } { p } { t } { b } }
2390     { #1 }
2391     {

```

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.

```

2392         \str_if_eq:eeTF { \l_@@_hpos_col_str } { j }
2393         { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2394     {

```

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

```

2395         \def \exp_not:N \l_@@_hpos_cell_tl
2396         { \str_lowercase:f { \l_@@_hpos_col_str } }
2397     }
2398     \IfPackageLoadedTF { ragged2e }
2399     {
2400         \str_case:on \l_@@_hpos_col_str
2401         {

```

The following `\exp_not:N` are mandatory.

```

2402             c { \exp_not:N \Centering }
2403             l { \exp_not:N \RaggedRight }
2404             r { \exp_not:N \RaggedLeft }
2405         }
2406     }
2407     {
2408         \str_case:on \l_@@_hpos_col_str
2409         {
2410             c { \exp_not:N \centering }
2411             l { \exp_not:N \raggedright }
2412             r { \exp_not:N \raggedleft }
2413         }
2414     }
2415     #3
2416 }
2417 { \str_if_eq:eeT { \l_@@_vpos_col_str } { m } \@@_center_cell_box: }
2418 { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_begin:w }
2419 { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_end: }
2420 { #2 }
2421 {
2422     \str_case:onF \l_@@_hpos_col_str
2423     {
2424         { j } { c }
2425         { si } { c }
2426     }

```

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

```

2427         { \str_lowercase:f \l_@@_hpos_col_str }
2428     }
2429 }

```

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

```

2430     \int_gincr:N \c@jCol
2431     \@@_rec_preamble_after_col:n
2432 }

```

#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*.

```

2433 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2434 {
2435   \str_if_eq:eeTF { \l_@@_hpos_col_str } { si }
2436   {
2437     \tl_gput_right:Nn \g_@@_array_preamble_tl
2438     { > \@@_test_if_empty_for_S: }
2439   }
2440   { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2441   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2442   \tl_gclear:N \g_@@_pre_cell_tl
2443   \tl_gput_right:Nn \g_@@_array_preamble_tl
2444   {
2445     > {

```

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.

```

2446       \dim_set:Nn \l_@@_col_width_dim { #2 }
2447       \@@_cell_begin:

```

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

```

2448       \use:c { #7 } [ #1 ] { #2 }

```

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

```

2449       \everypar
2450       {
2451         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2452         \everypar { }
2453       }

```

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

```

2454       #3

```

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

```

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

```

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

```

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

```

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

```

2464       #4
2465       \@@_cell_end:
2466     }
2467   }
2468 }

```

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

```

2469 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2470 {

```

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 `&`).

```

2471 \group_align_safe_begin:
2472 \peek_meaning:NTF &
2473 { \@@_the_cell_is_empty: }
2474 {
2475   \peek_meaning:NTF \\\
2476   { \@@_the_cell_is_empty: }
2477   {
2478     \peek_meaning:NTF \crcr
2479     \@@_the_cell_is_empty:
2480     \group_align_safe_end:
2481   }
2482 }
2483 }
2484 \cs_new_protected:Npn \@@_the_cell_is_empty:
2485 {
2486   \group_align_safe_end:
2487   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2488   {

```

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

```

2489   \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2490   \skip_horizontal:N \l_@@_col_width_dim
2491 }
2492 }
2493 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2494 {
2495   \peek_meaning:NT \__siunitx_table_skip:n
2496   { \bool_gset_true:N \g_@@_empty_cell_bool }
2497 }

```

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.

```

2498 \cs_new_protected:Npn \@@_center_cell_box:
2499 {

```

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

```

2500   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2501   {
2502     \dim_compare:nNnT
2503     { \box_ht:N \l_@@_cell_box }
2504     >

```

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).

```

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

```

```

2513         }
2514         { \box_use:N \l_@@_cell_box }
2515     }
2516 }
2517 }
2518 }

```

For V (similar to the V of varwidth).

```

2519 \cs_new_protected:Npn \@@_V: #1 #2
2520 {
2521     \str_if_eq:nnTF { #2 } { [ ] }
2522     { \@@_make_preamble_V_i:w [ ] }
2523     { \@@_make_preamble_V_i:w [ ] { #2 } }
2524 }
2525 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2526 { \@@_make_preamble_V_ii:nn { #1 } }
2527 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2528 {
2529     \str_set:Nn \l_@@_vpos_col_str { p }
2530     \str_set:Nn \l_@@_hpos_col_str { j }
2531     \@@_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`.

```

2532     \setlength { \l_tmpa_dim } { #2 }
2533     \IfPackageLoadedTF { varwidth }
2534     { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2535     {
2536         \@@_error_or_warning:n { varwidth-not-loaded }
2537         \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2538     }
2539 }

```

For w and W

```

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

```

#1 is a special argument: empty for w and equal to `\@@_special_W:` for W;

#2 is the type of column (w or W);

#3 is the type of horizontal alignment (c, l, r or s);

#4 is the width of the column.

```

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

```

First, the case of an horizontal alignment equal to s (for *stretch*).

#1 is a special argument: empty for w and equal to `\@@_special_W:` for W;

#2 is the width of the column.

```

2548 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2549 {
2550     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2551     \tl_gclear:N \g_@@_pre_cell_tl
2552     \tl_gput_right:Nn \g_@@_array_preamble_tl
2553     {
2554         > {

```

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`.

```

2555         \setlength { \l_@@_col_width_dim } { #2 }
2556         \@@_cell_begin:
2557         \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2558     }
2559     c
2560     < {
2561         \@@_cell_end_for_w_s:
2562         #1
2563         \@@_adjust_size_box:
2564         \box_use_drop:N \l_@@_cell_box
2565     }
2566 }
2567 \int_gincr:N \c@jCol
2568 \@@_rec_preamble_after_col:n
2569 }

```

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

```

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

```

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

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`.

```

2577         \setlength { \l_@@_col_width_dim } { #4 }
2578         \hbox_set:Nw \l_@@_cell_box
2579         \@@_cell_begin:
2580         \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2581     }
2582     c
2583     < {
2584         \@@_cell_end:
2585         \hbox_set_end:
2586         #1
2587         \@@_adjust_size_box:
2588         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2589     }
2590 }

```

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

```

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

```

```

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

```

For `S` (of `siunitx`).

```

2599 \cs_new_protected:Npn \@@_S: #1 #2
2600 {

```

```

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

```

In the cells of a column of type S, we have to wrap the command `\@@_node_cell:` for the horizontal alignment of the content of the cell (siunitx has done a job but it's without effect since we have to put the content in a box for the PGF/TikZ node and that's why we have to do the job of horizontal alignment once again).

```

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

```

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

```

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

```

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

```

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

```

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

```

2637 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2638 {
2639 \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }

```

If we are before the column 1 and not in `{NiceArray}`, we reserve space for the left delimiter.

```

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

```

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

```

2644 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2645 \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl

```

```

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

```

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

```

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

```

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

```

2670 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2671 {
2672     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2673     \tl_if_in:nnTF { } ] \} { #2 }
2674     { \@@_make_preamble_v:nnn #1 #2 }
2675     {
2676         \str_if_eq:nnTF { \s_stop } { #2 }
2677         {
2678             \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2679             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2680             {
2681                 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2682                 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2683                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2684                 \@@_rec_preamble:n #2
2685             }
2686         }
2687         {
2688             \tl_if_in:nnT { ( [ \{ \left } { #2 }
2689             { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2690             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2691             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2692             \@@_rec_preamble:n #2
2693         }
2694     }
2695 }
2696 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2697 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2698 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2699 {
2700     \str_if_eq:nnTF { \s_stop } { #3 }

```



```

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

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

```

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

```

2725 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2726 {
2727   \str_if_eq:nnTF { #1 } { < }
2728   { \@@_rec_preamble_after_col_i:n }
2729   {
2730     \str_if_eq:nnTF { #1 } { @ }
2731     { \@@_rec_preamble_after_col_ii:n }
2732     {
2733       \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2734       {
2735         \tl_gput_right:Nn \g_@@_array_preamble_tl
2736         { ! { \skip_horizontal:N \arrayrulewidth } }
2737       }
2738       {
2739         \clist_if_in:NeT \l_@@_vlines_clist
2740         { \int_eval:n { \c@jCol + 1 } }
2741         {
2742           \tl_gput_right:Nn \g_@@_array_preamble_tl
2743           { ! { \skip_horizontal:N \arrayrulewidth } }
2744         }
2745       }
2746       \@@_rec_preamble:n { #1 }
2747     }
2748   }
2749 }

2750 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2751 {
2752   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2753   \@@_rec_preamble_after_col:n
2754 }

```

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

```

2755 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2756 {
2757   \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2758   {
2759     \tl_gput_right:Nn \g_@@_array_preamble_tl
2760     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2761   }
2762   {
2763     \clist_if_in:NneTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2764     {
2765       \tl_gput_right:Nn \g_@@_array_preamble_tl
2766       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2767     }
2768     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2769   }
2770   \@@_rec_preamble:n
2771 }

2772 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2773 {
2774   \tl_clear:N \l_tmpa_tl
2775   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2776   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2777 }

```

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

```

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

```

For the case of a letter X. This specifier may take in an optional argument (between square brackets). That's why we test whether there is a `[` after the letter X.

```

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

```

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

The following set of keys is for the specifier X in the preamble of the array. Such specifier may have as keys all the keys of `{ nicematrix / p-column }` but also a key 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`.

```

2788 \keys_define:nn { nicematrix / X-column }
2789 {
2790   V .code:n =
2791     \IfPackageLoadedTF { varwidth }
2792     {
2793       \bool_set_true:N \l_@@_V_of_X_bool
2794       \bool_gset_true:N \g_@@_V_of_X_bool
2795     }
2796     { \@@_error_or_warning:n { varwidth-not-loaded-in-X } } ,
2797   unknown .code:n =
2798     \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2799     { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2800     { \@@_error_or_warning:n { invalid-weight } }
2801 }

```

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

```
2802 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2803 {
```

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).

```
2804 \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).

```
2805 \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.

```
2806 \fp_set:Nn \l_tmpa_fp { 1.0 }
```

```
2807 \@@_keys_p_column:n { #1 }
```

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

```
2808 \bool_set_false:N \l_@@_V_of_X_bool
2809 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
```

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

```
2810 \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).

```
2811 \bool_if:NTF \l_@@_X_columns_aux_bool
2812 {
2813 \@@_make_preamble_ii_iv:nnn
```

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

```
2814 { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2815 { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
2816 { \@@_no_update_width: }
2817 }
```

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.

```
2818 {
2819 \tl_gput_right:Nn \g_@@_array_preamble_tl
2820 {
2821 > {
2822 \@@_cell_begin:
2823 \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.

```
2824 \NotEmpty
```

The following code will nullify the box of the cell.

```
2825 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2826 { \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.

```
2827 \begin { minipage } { 5 cm } \arraybackslash
2828 }
2829 c
2830 < {
2831 \end { minipage }
2832 \@@_cell_end:
```

```

2833     }
2834   }
2835   \int_gincr:N \c@jCol
2836   \@@_rec_preamble_after_col:n
2837 }
2838 }

2839 \cs_new_protected:Npn \@@_no_update_width:
2840 {
2841   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2842   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2843 }

```

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

```

2844 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2845 {
2846   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2847   { \int_eval:n { \c@jCol + 1 } }
2848   \tl_gput_right:Ne \g_@@_array_preamble_tl
2849   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2850   \@@_rec_preamble:n
2851 }

```

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.

```

2852 \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).

```

2853 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2854 { \@@_fatal:n { Preamble-forgotten } }
2855 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
2856 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
2857 { @@ _ \token_to_str:N \hline : }
2858 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
2859 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
2860 { @@ _ \token_to_str:N \hline : }
2861 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
2862 { @@ _ \token_to_str:N \hline : }
2863 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
2864 { @@ _ \token_to_str:N \hline : }
2865 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
2866 { @@ _ \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).

```

2867 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2868 {

```

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`.

```

2869   \multispan { #1 }
2870   \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2871   \begingroup
2872   \tbl_update_multicolumn_cell_data:n { #1 }

```

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

```
2873 \tl_gclear:N \g_@@_preamble_tl
2874 \@@_make_m_preamble:n #2 \q_stop
```

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

```
2875 \def \@addamp
2876 {
2877   \legacy_if:nTF { @firstamp }
2878   { \legacy_if_set_false:n { @firstamp } }
2879   { \@preamerr 5 }
2880 }
2881 \exp_args:No \@mkpream \g_@@_preamble_tl
2882 \@addtopreamble \@empty
2883 \endgroup
2884 \UseTaggingSocket { tbl / colspan } { #1 }
```

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

```
2885 \int_compare:nNtT { #1 } > { \c_one_int }
2886 {
2887   \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2888   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2889   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2890   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2891   {
2892     {
2893       \int_if_zero:nTF { \c@jCol }
2894       { \int_eval:n { \c@iRow + 1 } }
2895       { \int_use:N \c@iRow }
2896     }
2897     { \int_eval:n { \c@jCol + 1 } }
2898     {
2899       \int_if_zero:nTF { \c@jCol }
2900       { \int_eval:n { \c@iRow + 1 } }
2901       { \int_use:N \c@iRow }
2902     }
2903     { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block.

```
2904 { }
2905 }
2906 }
```

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

```
2907 \RenewDocumentCommand { \cellcolor } { 0 { } m }
2908 {
2909   \tl_gput_right:Ne \g_@@_pre_code_before_tl
2910   {
2911     \@_rectanglecolor [ ##1 ]
2912     { \exp_not:n { ##2 } }
2913     { \int_use:N \c@iRow - \int_use:N \c@jCol }
2914     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2915   }
2916   \ignorespaces
2917 }
```

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

```
2918 \def \@sharp { #3 }
2919 \@arstrut
2920 \@preamble
2921 \null
```

We add some lines.

```

2922 \int_gadd:Nn \c@jCol { #1 - 1 }
2923 \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
2924 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2925 \ignorespaces
2926 }

```

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`.

```

2927 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2928 {
2929   \str_case:nnF { #1 }
2930   {
2931     c { \@@_make_m_preamble_i:n #1 }
2932     l { \@@_make_m_preamble_i:n #1 }
2933     r { \@@_make_m_preamble_i:n #1 }
2934     > { \@@_make_m_preamble_ii:nn #1 }
2935     ! { \@@_make_m_preamble_ii:nn #1 }
2936     @ { \@@_make_m_preamble_ii:nn #1 }
2937     | { \@@_make_m_preamble_iii:n #1 }
2938     p { \@@_make_m_preamble_iv:nnn t #1 }
2939     m { \@@_make_m_preamble_iv:nnn c #1 }
2940     b { \@@_make_m_preamble_iv:nnn b #1 }
2941     w { \@@_make_m_preamble_v:nnnn { } #1 }
2942     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2943     \q_stop { }
2944   }
2945   {
2946     \cs_if_exist:cTF { NC @ find @ #1 }
2947     {
2948       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2949       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2950     }
2951     {
2952       \str_if_eq:nnTF { #1 } { S }
2953       { \@@_fatal:n { unknown~column~type~S~multicolumn } }
2954       { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
2955     }
2956   }
2957 }

```

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

```

2958 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2959 {
2960   \tl_gput_right:Nn \g_@@_preamble_tl
2961   {
2962     > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
2963     #1
2964     < \@@_cell_end:
2965   }

```

We test for the presence of a `<`.

```

2966   \@@_make_m_preamble_x:n
2967 }

```

For `>`, `!` and `@`

```

2968 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2969 {
2970   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2971   \@@_make_m_preamble:n
2972 }

```

For l

```

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

```

For p, m and b

```

2978 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2979 {
2980   \tl_gput_right:Nn \g_@@_preamble_tl
2981   {
2982     > {
2983       \@@_cell_begin:

```

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

```

2984       \setlength { \l_tmpa_dim } { #3 }
2985       \begin { minipage } [ #1 ] { \l_tmpa_dim }
2986       \mode_leave_vertical:
2987       \arraybackslash
2988       \vrule height \box_ht:N \@@arstrutbox depth \c_zero_dim width \c_zero_dim
2989     }
2990     c
2991     < {
2992       \vrule height \c_zero_dim depth \box_dp:N \@@arstrutbox width \c_zero_dim
2993       \end { minipage }
2994       \@@_cell_end:
2995     }
2996   }

```

We test for the presence of a `<`.

```

2997   \@@_make_m_preamble_x:n
2998 }

```

For w and W

```

2999 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3000 {
3001   \tl_gput_right:Nn \g_@@_preamble_tl
3002   {
3003     > {
3004       \dim_set:Nn \l_@@_col_width_dim { #4 }
3005       \hbox_set:Nw \l_@@_cell_box
3006       \@@_cell_begin:
3007       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3008     }
3009     c
3010     < {
3011       \@@_cell_end:
3012       \hbox_set_end:
3013       \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3014       #1
3015       \@@_adjust_size_box:
3016       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3017     }
3018   }

```

We test for the presence of a `<`.

```

3019   \@@_make_m_preamble_x:n
3020 }

```

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

```

3021 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3022 {
3023   \str_if_eq:nnTF { #1 } { < }
3024     { \@@_make_m_preamble_ix:n }
3025     { \@@_make_m_preamble:n { #1 } }
3026 }
3027 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3028 {
3029   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3030   \@@_make_m_preamble_x:n
3031 }

```

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`).

```

3032 \cs_new_protected:Npn \@@_put_box_in_flow:
3033 {
3034   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3035   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3036   \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
3037     { \box_use_drop:N \l_tmpa_box }
3038     { \@@_put_box_in_flow_i: }
3039 }

```

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

```

3040 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3041 {
3042   \pgfpicture
3043     \@@_qpoint:n { row - 1 }
3044     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3045     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3046     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3047     \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).

```

3048   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3049   {
3050     \int_set:Nn \l_tmpa_int
3051       { \str_range:Nnn \l_@@_baseline_tl { 6 } { -1 } }
3052     \bool_lazy_or:nnT
3053       { \int_compare_p:nNn { \l_tmpa_int } < { 1 } }
3054       { \int_compare_p:nNn { \l_tmpa_int } > { \c@iRow + 1 } }
3055     {
3056       \@@_error:n { bad-value-for-baseline-line }
3057       \int_set_eq:NN \l_tmpa_int \c_one_int
3058     }
3059     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3060   }
3061   {
3062     \str_if_eq:eeTF { \l_@@_baseline_tl } { t }
3063     { \int_set_eq:NN \l_tmpa_int \c_one_int }
3064     {
3065       \str_if_eq:onTF \l_@@_baseline_tl { b }
3066       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3067       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3068     }
3069     \bool_lazy_or:nnT

```



```

3070      { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3071      { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3072      {
3073          \@@_error:n { bad-value~for~baseline }
3074          \int_set_eq:NN \l_tmpa_int \c_one_int
3075      }
3076      \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3077      \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3078      }
3079      \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3080      \endpgfpicture
3081      \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3082      \box_use_drop:N \l_tmpa_box
3083  }

```

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).

```

3084 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3085 {

```

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.

```

3086     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3087     {
3088         \int_compare:nNnT { \c@jCol } > { \c_one_int }
3089         {
3090             \box_set_wd:Nn \l_@@_the_array_box
3091             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3092         }
3093     }

```

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

```

3094     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3095     \bool_if:NT \l_@@_caption_above_bool
3096     {
3097         \tl_if_empty:NF \l_@@_caption_tl
3098         {
3099             \bool_set_false:N \g_@@_caption_finished_bool
3100             \int_gzero:N \c@tabularnote
3101             \@@_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).

```

3102             \int_compare:nNnT { \g_@@_notes_caption_int } > { \c_zero_int }
3103             {
3104                 \tl_gput_right:Ne \g_@@_aux_tl
3105                 {
3106                     \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3107                     { \int_use:N \g_@@_notes_caption_int }
3108                 }
3109                 \int_gzero:N \g_@@_notes_caption_int
3110             }
3111         }
3112     }

```

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

```

3113   \hbox
3114   {
3115     \box_use_drop:N \l_@@_the_array_box

```

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

```

3116     \@@_create_extra_nodes:
3117     \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3118   }

```

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).

```

3119   \bool_lazy_any:nT
3120   {
3121     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3122     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3123     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3124   }
3125   \@@_insert_tabularnotes:
3126   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3127   \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3128   \end { minipage }
3129 }

```

```

3130 \cs_new_protected:Npn \@@_insert_caption:
3131 {
3132   \tl_if_empty:NF \l_@@_caption_tl
3133   {
3134     \cs_if_exist:NTF \@capttype
3135     { \@@_insert_caption_i: }
3136     { \@@_error:n { caption~outside~float } }
3137   }
3138 }

```

```

3139 \cs_new_protected:Npn \@@_insert_caption_i:
3140 {
3141   \group_begin:

```

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

```

3142   \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.

```

3143   \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
3144   \tl_if_empty:NTF \l_@@_short_caption_tl
3145   { \caption }
3146   { \caption [ \l_@@_short_caption_tl ] }
3147   { \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.

```

3148   \bool_if:NF \g_@@_caption_finished_bool

```

```

3149     {
3150         \bool_gset_true:N \g_@@_caption_finished_bool
3151         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3152         \int_gzero:N \c@tabularnote
3153     }
3154     \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3155     \group_end:
3156 }

3157 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3158 {
3159     \@@_error_or_warning:n { tabularnote~below~the~tabular }
3160     \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3161 }

3162 \cs_new_protected:Npn \@@_insert_tabularnotes:
3163 {
3164     \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3165     \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3166     \skip_vertical:N 0.65ex

```

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

```

3167     \group_begin:
3168     \l_@@_notes_code_before_tl
3169     \tl_if_empty:NF \g_@@_tabularnote_tl
3170     {
3171         \g_@@_tabularnote_tl \par
3172         \tl_gclear:N \g_@@_tabularnote_tl
3173     }

```

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.

```

3174     \int_compare:nNnT { \c@tabularnote } > { \c_zero_int }
3175     {
3176         \bool_if:NTF \l_@@_notes_para_bool
3177         {
3178             \begin { tabularnotes* }
3179             \seq_map_inline:Nn \g_@@_notes_seq
3180             { \@@_one_tabularnote:nn ##1 }
3181             \strut
3182             \end { tabularnotes* }

```

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

```

3183         \par
3184     }
3185     {
3186         \tabularnotes
3187         \seq_map_inline:Nn \g_@@_notes_seq
3188         { \@@_one_tabularnote:nn ##1 }
3189         \strut
3190         \endtabularnotes
3191     }
3192 }
3193 \unskip
3194 \group_end:
3195 \bool_if:NT \l_@@_notes_bottomrule_bool
3196 {
3197     \IfPackageLoadedTF { booktabs }
3198     {

```

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

```

3199         \skip_vertical:N \aboverulesep

```

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

```

3200         { \CT@arc@ \hrule height \heavyrulewidth }

```

```

3201     }
3202     { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3203   }
3204   \l_@@_notes_code_after_tl
3205   \seq_gclear:N \g_@@_notes_seq
3206   \seq_gclear:N \g_@@_notes_in_caption_seq
3207   \int_gzero:N \c@tabularnote
3208 }

```

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.

```

3209 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3210 {
3211   \tl_if_novalue:nTF { #1 }
3212     { \item }
3213     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3214 }

```

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.

```

3215 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3216 {
3217   \pgfpicture
3218     \@@_qpoint:n { row - 1 }
3219     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3220     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3221     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3222   \endpgfpicture
3223   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3224   \int_if_zero:nT { \l_@@_first_row_int }
3225   {
3226     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3227     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3228   }
3229   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3230 }

```

Now, the general case.

```

3231 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3232 {

```

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

```

3233   \str_if_eq:eeT { \l_@@_baseline_tl } { t }
3234   { \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`.

```

3235   \pgfpicture
3236     \@@_qpoint:n { row - 1 }
3237     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3238     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3239     {
3240       \int_set:Nn \l_tmpa_int
3241       {
3242         \str_range:Nnn
3243           \l_@@_baseline_tl
3244           { 6 }
3245           { \tl_count:o \l_@@_baseline_tl }
3246       }
3247     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }

```

```

3248 }
3249 {
3250   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3251   \bool_lazy_or:nnT
3252     { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3253     { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3254     {
3255       \@@_error:n { bad-value-for-baseline }
3256       \int_set:Nn \l_tmpa_int 1
3257     }
3258   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3259 }
3260 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3261 \endpgfpicture
3262 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3263 \int_if_zero:nT { \l_@@_first_row_int }
3264 {
3265   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3266   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3267 }
3268 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3269 }

```

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.

```

3270 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3271 {

```

We will compute the real width of both delimiters used.

```

3272   \dim_zero_new:N \l_@@_real_left_delim_dim
3273   \dim_zero_new:N \l_@@_real_right_delim_dim
3274   \hbox_set:Nn \l_tmpb_box
3275   {
3276     \m@th
3277     $ % $
3278     \left #1
3279     \vcenter
3280     {
3281       \vbox_to_ht:nn
3282         { \box_ht_plus_dp:N \l_tmpa_box }
3283       { }
3284     }
3285     \right .
3286     $ % $
3287   }
3288   \dim_set:Nn \l_@@_real_left_delim_dim
3289     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3290   \hbox_set:Nn \l_tmpb_box
3291   {
3292     \m@th
3293     $ % $
3294     \left .
3295     \vbox_to_ht:nn
3296       { \box_ht_plus_dp:N \l_tmpa_box }
3297     { }
3298     \right #2
3299     $ % $
3300   }
3301   \dim_set:Nn \l_@@_real_right_delim_dim
3302     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3303   \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }

```

```

3304 \@@_put_box_in_flow:
3305 \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3306 }

```

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).

```

3307 \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).

```

3308 {
3309   \peek_remove_spaces:n
3310   {
3311     \peek_meaning:NTF \end
3312     { \@@_analyze_end:Nn }
3313     {
3314       \@@_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`).

```

3315       \@@_array:o \g_@@_array_preamble_tl
3316     }
3317   }
3318 }
3319 {
3320   \@@_create_col_nodes:
3321   \endarray
3322 }

```

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

```

3323 \NewDocumentEnvironment { @@-light-syntax } { b }
3324 {

```

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`.

```

3325 \tl_if_empty:nT { #1 }
3326 { \@@_fatal:n { empty-environment } }
3327 \tl_if_in:nnT { #1 } { & }
3328 { \@@_fatal:n { ampersand-in~light-syntax } }
3329 \tl_if_in:nnT { #1 } { \ }
3330 { \@@_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`.

```

3331 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3332 }

```

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.

```

3333 {

```

```

3334 \@@_create_col_nodes:
3335 \endarray
3336 }
3337 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3338 {
3339 \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).

```

3340 \seq_clear_new:N \l_@@_rows_seq

```

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

```

3341 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3342 \bool_if:NTF \l_@@_light_syntax_expanded_bool
3343 { \seq_set_split:Nee }
3344 { \seq_set_split:Non }
3345 \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3346 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3347 \tl_if_empty:NF \l_tmpa_tl
3348 { \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.

```

3349 \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
3350 { \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`).

```

3351 \tl_build_begin:N \l_@@_new_body_tl
3352 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3353 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3354 \@@_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).

```

3355 \seq_map_inline:Nn \l_@@_rows_seq
3356 {
3357 \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3358 \@@_line_with_light_syntax:n { ##1 }
3359 }
3360 \tl_build_end:N \l_@@_new_body_tl
3361 \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
3362 {
3363 \int_set:Nn \l_@@_last_col_int
3364 { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3365 }

```

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.

```

3366 \@@_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`).

```

3367 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3368 }

```

```

3369 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3370 {
3371   \seq_clear_new:N \l_@@_cells_seq
3372   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3373   \int_set:Nn \l_@@_nb_cols_int
3374   {
3375     \int_max:nn
3376     { \l_@@_nb_cols_int }
3377     { \seq_count:N \l_@@_cells_seq }
3378   }
3379   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3380   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3381   \seq_map_inline:Nn \l_@@_cells_seq
3382   { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3383 }
3384 \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`.

```

3385 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3386 {
3387   \str_if_eq:eeT { \g_@@_name_env_str } { #2 }
3388   { \@@_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.

```

3389   \end { #2 }
3390 }

```

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`).

```

3391 \cs_new:Npn \@@_create_col_nodes:
3392 {
3393   \crrc
3394   \int_if_zero:nT { \l_@@_first_col_int }
3395   {
3396     \omit
3397     \hbox_overlap_left:n
3398     {
3399       \bool_if:NT \l_@@_code_before_bool
3400       { \pgfsys@markposition { \@@_env: - col - 0 } }
3401       \pgfpicture
3402       \pgfrememberpicturepositiononpagetrue
3403       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3404       \str_if_empty:NF \l_@@_name_str
3405       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3406       \endpgfpicture
3407       \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3408     }
3409     &
3410   }
3411   \omit

```

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

```

3412   \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`).

```

3413   \int_if_zero:nTF { \l_@@_first_col_int }
3414   {
3415     \@@_mark_position:n { 1 }

```



```

3416 \pgfpicture
3417 \pgfrememberpicturepositiononpagetrue
3418 \pgfcoordinate { \@@_env: - col - 1 }
3419 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3420 \str_if_empty:NF \l_@@_name_str
3421 { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3422 \endpgfpicture
3423 }
3424 {
3425 \bool_if:NT \l_@@_code_before_bool
3426 {
3427 \hbox
3428 {
3429 \skip_horizontal:n { 0.5 \arrayrulewidth }
3430 \pgfsys@markposition { \@@_env: - col - 1 }
3431 \skip_horizontal:n { -0.5 \arrayrulewidth }
3432 }
3433 }
3434 \pgfpicture
3435 \pgfrememberpicturepositiononpagetrue
3436 \pgfcoordinate { \@@_env: - col - 1 }
3437 { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3438 \@@_node_alias:n { 1 }
3439 \endpgfpicture
3440 }

```

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.

```

3441 \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
3442 \bool_if:NF \l_@@_auto_columns_width_bool
3443 { \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim } }
3444 {
3445 \bool_lazy_and:nnTF
3446 { \l_@@_auto_columns_width_bool }
3447 { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3448 { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3449 { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3450 \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3451 }
3452 \skip_horizontal:N \g_tmpa_skip
3453 \hbox
3454 {
3455 \@@_mark_position:n { 2 }
3456 \pgfpicture
3457 \pgfrememberpicturepositiononpagetrue
3458 \pgfcoordinate { \@@_env: - col - 2 }
3459 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3460 \@@_node_alias:n { 2 }
3461 \endpgfpicture
3462 }

```

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.

```

3463 \int_gset_eq:NN \g_tmpa_int \c_one_int
3464 \bool_if:NTF \g_@@_last_col_found_bool
3465 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3466 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3467 {
3468 &
3469 \omit

```

```
3470 \int_gincr:N \g_tmpa_int
```

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

```
3471 \skip_horizontal:N \g_tmpa_skip
3472 \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }
```

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

```
3473 \pgfpicture
3474 \pgfrememberpicturepositiononpagetrue
3475 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3476 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3477 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3478 \endpgfpicture
3479 }
```

```
3480 % & % moved on 2025-11-02
3481 % \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).

```
3482 \bool_lazy_or:nnF
3483 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3484 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3485 {
3486   &
3487   \omit
3488   \skip_horizontal:N \g_tmpa_skip
3489   \int_gincr:N \g_tmpa_int
3490   \bool_lazy_any:nF
3491   {
3492     \g_@@_delims_bool
3493     \l_@@_tabular_bool
3494     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3495     \l_@@_exterior_arraycolsep_bool
3496     \l_@@_bar_at_end_of_pream_bool
3497   }
3498   { \skip_horizontal:n { - \col@sep } }
3499   \bool_if:NT \l_@@_code_before_bool
3500   {
3501     \hbox
3502     {
3503       \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.

```
3504 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3505 { \skip_horizontal:n { - \arraycolsep } }
3506 \pgfsys@markposition
3507 { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3508 \skip_horizontal:n { 0.5 \arrayrulewidth }
3509 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3510 { \skip_horizontal:N \arraycolsep }
3511 }
3512 }
3513 \pgfpicture
3514 \pgfrememberpicturepositiononpagetrue
3515 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3516 {
3517   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3518   {
3519     \pgfpoint
3520     { - 0.5 \arrayrulewidth - \arraycolsep }
3521     \c_zero_dim
```

```

3522     }
3523     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3524   }
3525   \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3526   \endpgfpicture
3527 }

3528 \bool_if:NT \g_@@_last_col_found_bool
3529 {
3530   \hbox_overlap_right:n
3531   {
3532     \skip_horizontal:N \g_@@_width_last_col_dim
3533     \skip_horizontal:N \col@sep
3534     \bool_if:NT \l_@@_code_before_bool
3535     {
3536       \pgfsys@markposition
3537       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3538     }
3539     \pgfpicture
3540     \pgfrememberpicturepositiononpagetrue
3541     \pgfcoordinate
3542     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3543     \pgfpointorigin
3544     \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3545     \endpgfpicture
3546   }
3547 }
3548 }

3549 \cs_new_protected:Npn \@@_mark_position:n #1
3550 {
3551   \bool_if:NT \l_@@_code_before_bool
3552   {
3553     \hbox
3554     {
3555       \skip_horizontal:n { -0.5 \arrayrulewidth }
3556       \pgfsys@markposition { \@@_env: - col - #1 }
3557       \skip_horizontal:n { 0.5 \arrayrulewidth }
3558     }
3559   }
3560 }

3561 \cs_new_protected:Npn \@@_node_alias:n #1
3562 {
3563   \str_if_empty:NF \l_@@_name_str
3564   { \pgfnodealias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3565 }

```

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

```

3566 \tl_const:Nn \c_@@_preamble_first_col_tl
3567 {
3568   >
3569   {

```

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

```

3570     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3571     \bool_gset_true:N \g_@@_after_col_zero_bool
3572     \@@_begin_of_row:
3573     \hbox_set:Nw \l_@@_cell_box
3574     \@@_math_toggle:
3575     \@@_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”.

```

3576     \int_compare:nNt { \c@iRow } > { \c_zero_int }
3577     {
3578         \bool_lazy_or:nnT
3579         { \int_compare_p:nN { \l_@@_last_row_int } < { \c_zero_int } }
3580         { \int_compare_p:nN { \c@iRow } < { \l_@@_last_row_int } }
3581         {
3582             \l_@@_code_for_first_col_tl
3583             \xglobal \colorlet { nicematrix-first-col } { . }
3584         }
3585     }
3586 }

```

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`.

```

3587     l
3588     <
3589     {
3590         \@@_math_toggle:
3591         \hbox_set_end:
3592         \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3593         \@@_adjust_size_box:
3594         \@@_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.

```

3595     \dim_gset:Nn \g_@@_width_first_col_dim
3596     { \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.

```

3597     \hbox_overlap_left:n
3598     {
3599         \dim_compare:nNtF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3600         { \@@_node_cell: }
3601         { \box_use_drop:N \l_@@_cell_box }
3602         \skip_horizontal:N \l_@@_left_delim_dim
3603         \skip_horizontal:N \l_@@_left_margin_dim
3604         \skip_horizontal:N \l_@@_extra_left_margin_dim
3605     }
3606     \bool_gset_false:N \g_@@_empty_cell_bool
3607     \skip_horizontal:n { -2 \col@sep }
3608 }
3609 }

```

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

```

3610 \tl_const:Nn \c_@@_preamble_last_col_tl
3611 {
3612     >
3613     {
3614         \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).

```

3615     \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.

```

3616     \bool_gset_true:N \g_@@_last_col_found_bool
3617     \int_gincr:N \c@jCol
3618     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3619     \hbox_set:Nw \l_@@_cell_box
3620     \@@_math_toggle:
3621     \@@_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”.

```

3622     \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3623     {
3624         \bool_lazy_or:nnT
3625         { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3626         { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3627         {
3628             \l_@@_code_for_last_col_tl
3629             \xglobal \colorlet { nicematrix-last-col } { . }
3630         }
3631     }
3632 }
3633 1
3634 <
3635 {
3636     \@@_math_toggle:
3637     \hbox_set_end:
3638     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3639     \@@_adjust_size_box:
3640     \@@_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.

```

3641     \dim_gset:Nn \g_@@_width_last_col_dim
3642     { \dim_max:nn { \g_@@_width_last_col_dim } { \box_wd:N \l_@@_cell_box } }
3643     \skip_horizontal:n { -2 \col@sep }

```

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

```

3644     \hbox_overlap_right:n
3645     {
3646         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3647         {
3648             \skip_horizontal:N \l_@@_right_delim_dim
3649             \skip_horizontal:N \l_@@_right_margin_dim
3650             \skip_horizontal:N \l_@@_extra_right_margin_dim
3651             \@@_node_cell:
3652         }
3653     }
3654     \bool_gset_false:N \g_@@_empty_cell_bool
3655 }
3656 }

```

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

```

3657 \NewDocumentEnvironment { NiceArray } { }
3658 {
3659     \bool_gset_false:N \g_@@_delims_bool
3660     \str_if_empty:NT \g_@@_name_env_str
3661     { \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).

```

3662     \NiceArrayWithDelims . .
3663 }
3664 { \endNiceArrayWithDelims }

```

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

```

3665 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3666 {
3667     \NewDocumentEnvironment { #1 NiceArray } { }
3668     {

```

```

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

```

13 The environment `{NiceMatrix}` and its variants

```

3682 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3683 {
3684   \bool_set_false:N \l_@@_preamble_bool
3685   \tl_clear:N \l_tmpa_tl
3686   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3687   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3688   \tl_put_right:Nn \l_tmpa_tl
3689   {
3690     *
3691     {
3692       \int_case:nnF \l_@@_last_col_int
3693       {
3694         { -2 } { \c@MaxMatrixCols }
3695         { -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).

```

3696       }
3697       { \int_eval:n { \l_@@_last_col_int - 1 } }
3698     }
3699     { #2 }
3700   }
3701   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3702   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3703 }
3704 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3705 \clist_map_inline:nn { p , b , B , v , V }
3706 {
3707   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3708   {
3709     \bool_gset_true:N \g_@@_delims_bool
3710     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3711     \int_if_zero:nT { \l_@@_last_col_int }
3712     {
3713       \bool_set_true:N \l_@@_last_col_without_value_bool
3714       \int_set:Nn \l_@@_last_col_int { -1 }
3715     }
3716     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3717     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3718   }
3719   { \use:c { end #1 NiceArray } }
3720 }

```

We define also an environment `{NiceMatrix}`

```

3721 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3722 {
3723   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3724   \int_if_zero:nT { \l_@@_last_col_int }
3725   {
3726     \bool_set_true:N \l_@@_last_col_without_value_bool
3727     \int_set:Nn \l_@@_last_col_int { -1 }
3728   }
3729   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3730   \bool_lazy_or:nnT
3731     { \clist_if_empty_p:N \l_@@_vlines_clist }
3732     { \l_@@_except_borders_bool }
3733     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3734   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3735 }
3736 { \endNiceArray }

```

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

```

3737 \cs_new_protected:Npn \@@_NotEmpty:
3738 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3739 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3740 {

```

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.

```

3741   \dim_compare:nNnT { \l_@@_width_dim } = { \c_zero_dim }
3742     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3743   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3744   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3745   \tl_if_empty:NF \l_@@_short_caption_tl
3746   {
3747     \tl_if_empty:NT \l_@@_caption_tl
3748     {
3749       \@@_error_or_warning:n { short-caption-without-caption }
3750       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3751     }
3752   }
3753   \tl_if_empty:NF \l_@@_label_tl
3754   {
3755     \tl_if_empty:NT \l_@@_caption_tl
3756     { \@@_error_or_warning:n { label-without-caption } }
3757   }
3758   \NewDocumentEnvironment { TabularNote } { b }
3759   {
3760     \bool_if:NTF \l_@@_in_code_after_bool
3761       { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3762       {
3763         \tl_if_empty:NF \g_@@_tabularnote_tl
3764         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3765         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3766       }
3767   }
3768   { }
3769   \@@_settings_for_tabular:
3770   \NiceArray { #2 }
3771 }
3772 { \endNiceArray }
3773 \cs_new_protected:Npn \@@_settings_for_tabular:
3774 {

```

```

3775 \bool_set_true:N \l_@@_tabular_bool
3776 \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3777 \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3778 \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3779 }

3780 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3781 {
3782   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3783   \dim_set:Nn \l_@@_width_dim { #1 }
3784   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3785   \@@_settings_for_tabular:
3786   \NiceArray { #3 }
3787 }
3788 {
3789   \endNiceArray
3790   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3791     { \@@_error:n { NiceTabularX~without~X } }
3792 }

3793 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3794 {
3795   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3796   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3797   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3798   \@@_settings_for_tabular:
3799   \NiceArray { #3 }
3800 }
3801 { \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`).

```

3802 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3803 {
3804   \bool_lazy_all:nT
3805   {
3806     { \dim_compare_p:nNn { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim } }
3807     { \l_@@_hvlines_bool }
3808     { ! \g_@@_delims_bool }
3809     { ! \l_@@_except_borders_bool }
3810   }
3811   {
3812     \bool_set_true:N \l_@@_except_borders_bool
3813     \clist_if_empty:NF \l_@@_corners_clist
3814     { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3815     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3816     {
3817       \@@_stroke_block:nnn
3818       {
3819         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3820         draw = \l_@@_rules_color_tl
3821       }
3822       { 1-1 }
3823       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3824     }
3825   }
3826 }

```



```

3827 \cs_new_protected:Npn \@@_after_array:
3828 {

```

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

```

3829 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3830 \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.

```

3831 \bool_if:NT \g_@@_last_col_found_bool
3832 { \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`.

```

3833 \bool_if:NT \l_@@_last_col_without_value_bool
3834 { \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.

```

3835 \bool_if:NT \l_@@_last_row_without_value_bool
3836 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3837 \tl_gput_right:Ne \g_@@_aux_tl
3838 {
3839 \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3840 {
3841 \int_use:N \l_@@_first_row_int ,
3842 \int_use:N \c@iRow ,
3843 \int_use:N \g_@@_row_total_int ,
3844 \int_use:N \l_@@_first_col_int ,
3845 \int_use:N \c@jCol ,
3846 \int_use:N \g_@@_col_total_int
3847 }
3848 }

```

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`).

```

3849 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3850 {
3851 \tl_gput_right:Ne \g_@@_aux_tl
3852 {
3853 \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3854 { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3855 }
3856 }
3857 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3858 {
3859 \tl_gput_right:Ne \g_@@_aux_tl
3860 {
3861 \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3862 { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3863 \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3864 { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3865 }
3866 }

```

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

```
3867 \@@_create_diag_nodes:
```

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

```
3868 \pgfpicture
3869 \@@_create_aliases_last:
3870 \str_if_empty:NF \l_@@_name_str { \@@_create_alias_nodes: }
3871 \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.

```
3872 \bool_if:NT \l_@@_parallelize_diags_bool
3873 {
3874   \int_gzero:N \g_@@_ddots_int
3875   \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.

```
3876   \dim_gzero:N \g_@@_delta_x_one_dim
3877   \dim_gzero:N \g_@@_delta_y_one_dim
3878   \dim_gzero:N \g_@@_delta_x_two_dim
3879   \dim_gzero:N \g_@@_delta_y_two_dim
3880 }
3881 \bool_set_false:N \l_@@_initial_open_bool
3882 \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.

```
3883 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }
```

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

```
3884 \@@_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.

```
3885 \clist_if_empty:NF \l_@@_corners_clist
3886 {
3887   \bool_if:NTF \l_@@_no_cell_nodes_bool
3888   { \@@_error:n { corners~with~no~cell~nodes } }
3889   { \@@_compute_corners: }
3890 }
```

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

```
3891 \@@_adjust_pos_of_blocks_seq:
3892 \@@_deal_with_rounded_corners:
3893 \clist_if_empty:NF \l_@@_hlines_clist { \@@_draw_hlines: }
3894 \clist_if_empty:NF \l_@@_vlines_clist { \@@_draw_vlines: }
```

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

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

```

3895 \IfPackageLoadedT { tikz }
3896 {
3897   \tikzset
3898   {
3899     every-picture / .style =
3900     {
3901       overlay ,
3902       remember-picture ,
3903       name-prefix = \@@_env: -
3904     }
3905   }
3906 }
3907 \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
3908 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3909 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3910 \cs_set_eq:NN \OverBrace \@@_OverBrace
3911 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3912 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3913 \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.

```

3914 \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
3915 \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.

```

3916 \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).

```

3917 \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`).

```

3918 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3919 { \@@_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:`.

```

3920 \bool_set_true:N \l_@@_in_code_after_bool
3921 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3922 \scan_stop:
3923 \tl_gclear:N \g_nicematrix_code_after_tl
3924 \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.

```

3925 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3926 \tl_if_empty:NF \g_@@_pre_code_before_tl
3927 {
3928   \tl_gput_right:Ne \g_@@_aux_tl
3929   {
3930     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3931     { \exp_not:o \g_@@_pre_code_before_tl }
3932   }
3933   \tl_gclear:N \g_@@_pre_code_before_tl
3934 }

```

```

3935 \tl_if_empty:NF \g_nicematrix_code_before_tl
3936 {
3937   \tl_gput_right:Ne \g_@@_aux_tl
3938   {
3939     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3940     { \exp_not:o \g_nicematrix_code_before_tl }
3941   }
3942   \tl_gclear:N \g_nicematrix_code_before_tl
3943 }

3944 \str_gclear:N \g_@@_name_env_str
3945 \@@_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`.

```

3946 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3947 }

3948 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
3949 {
3950   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3951   \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }

3952   \dim_set:Nn \l_@@_xdots_shorten_start_dim
3953   { 0.6 \l_@@_xdots_shorten_start_dim }
3954   \dim_set:Nn \l_@@_xdots_shorten_end_dim
3955   { 0.6 \l_@@_xdots_shorten_end_dim }
3956 }

```

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.

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`.

```

3957 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3958 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

3959 \cs_new_protected:Npn \@@_create_alias_nodes:
3960 {
3961   \int_step_inline:nn { \c@iRow }
3962   {
3963     \pgfnodealias
3964     { \l_@@_name_str - ##1 - last }
3965     { \@@_env: - ##1 - \int_use:N \c@jCol }
3966   }
3967   \int_step_inline:nn { \c@jCol }
3968   {
3969     \pgfnodealias
3970     { \l_@@_name_str - last - ##1 }
3971     { \@@_env: - \int_use:N \c@iRow - ##1 }
3972   }
3973   \pgfnodealias
3974   { \l_@@_name_str - last - last }
3975   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
3976 }

```

¹³e.g. `\color[rgb]{0.5,0.5,0}`

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

```

3977 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3978 {
3979   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3980   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3981 }

```

The following command must *not* be protected.

```

3982 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3983 {
3984   { #1 }
3985   { #2 }
3986   {
3987     \int_compare:nNnTF { #3 } > { 98 }
3988     { \int_use:N \c@iRow }
3989     { #3 }
3990   }
3991   {
3992     \int_compare:nNnTF { #4 } > { 98 }
3993     { \int_use:N \c@jCol }
3994     { #4 }
3995   }
3996   { #5 }
3997 }

```

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).

```

3998 \hook_gput_code:nnn { begindocument } { . }
3999 {
4000   \cs_new_protected:Npe \@@_draw_dotted_lines:
4001   {
4002     \c_@@_pgfortikzpicture_tl
4003     \@@_draw_dotted_lines_i:
4004     \c_@@_endpgfortikzpicture_tl
4005   }
4006 }

```

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

```

4007 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4008 {
4009   \pgfrememberpicturepositiononpagetrue
4010   \pgf@relevantforpicturesizefalse
4011   \g_@@_HVdotsfor_lines_tl
4012   \g_@@_Vdots_lines_tl
4013   \g_@@_Ddots_lines_tl
4014   \g_@@_Idots_lines_tl
4015   \g_@@_Cdots_lines_tl
4016   \g_@@_Ldots_lines_tl
4017 }

4018 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4019 {
4020   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4021   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4022 }

```

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

```

4023 \pgfdeclareshape { @@_diag_node }
4024 {
4025   \savedanchor { \five }
4026   {
4027     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4028     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4029   }
4030   \anchor { 5 } { \five }
4031   \anchor { center } { \pgfpointorigin }
4032   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4033   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4034   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4035   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4036   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4037   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4038   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4039   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4040   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4041   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4042 }

```

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).

```

4043 \cs_new_protected:Npn \@@_create_diag_nodes:
4044 {
4045   \pgfpicture
4046   \pgfrememberpicturepositiononpagetrue
4047   \int_step_inline:nn { \int_max:nn { \c@iRow } { \c@jCol } }
4048   {
4049     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4050     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4051     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4052     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4053     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4054     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4055     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4056     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4057     \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.

```

4058     \dim_set:Nn \l_tmpa_int { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4059     \dim_set:Nn \l_tmpb_int { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4060     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4061     \str_if_empty:NF \l_@@_name_str
4062     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4063   }

```

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

```

4064     \int_set:Nn \l_tmpa_int { \int_max:nn { \c@iRow } { \c@jCol } + 1 }
4065     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4066     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4067     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4068     \pgfcoordinate
4069     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4070     \pgfnodealias
4071     { \@@_env: - last }
4072     { \@@_env: - \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
4073     \str_if_empty:NF \l_@@_name_str
4074     {

```

```

4075     \pgfnodealias
4076     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4077     { \@@_env: - \int_use:N \l_tmpa_int }
4078     \pgfnodealias
4079     { \l_@@_name_str - last }
4080     { \@@_env: - last }
4081   }
4082   \endpgfpicture
4083 }

```

16 We draw the dotted lines

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

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

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

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

This command computes:

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

```

4084 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4085 {

```

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

```

4086   \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4087   \int_set:Nn \l_@@_initial_i_int { #1 }
4088   \int_set:Nn \l_@@_initial_j_int { #2 }
4089   \int_set:Nn \l_@@_final_i_int { #1 }
4090   \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.

```

4091   \bool_set_false:N \l_@@_stop_loop_bool
4092   \bool_do_until:Nn \l_@@_stop_loop_bool
4093   {
4094     \int_add:Nn \l_@@_final_i_int { #3 }
4095     \int_add:Nn \l_@@_final_j_int { #4 }
4096     \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.

```

4097     \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4098     \if_int_compare:w #3 = \c_one_int
4099     \bool_set_true:N \l_@@_final_open_bool
4100   \else:
4101     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4102     \bool_set_true:N \l_@@_final_open_bool
4103   \fi:
4104 \fi:
4105 \else:
4106   \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4107   \if_int_compare:w #4 = -1
4108   \bool_set_true:N \l_@@_final_open_bool
4109   \fi:
4110 \else:
4111   \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4112   \if_int_compare:w #4 = \c_one_int
4113   \bool_set_true:N \l_@@_final_open_bool
4114   \fi:
4115 \fi:
4116 \fi:
4117 \fi:
4118 \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.

```

4119   {

```

We do a step backwards.

```

4120     \int_sub:Nn \l_@@_final_i_int { #3 }
4121     \int_sub:Nn \l_@@_final_j_int { #4 }
4122     \bool_set_true:N \l_@@_stop_loop_bool
4123   }

```

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`.

```

4124   {
4125     \cs_if_exist:cTF
4126     {
4127       @@ _ dotted _
4128       \int_use:N \l_@@_final_i_int -
4129       \int_use:N \l_@@_final_j_int
4130     }
4131     {
4132       \int_sub:Nn \l_@@_final_i_int { #3 }
4133       \int_sub:Nn \l_@@_final_j_int { #4 }
4134       \bool_set_true:N \l_@@_final_open_bool
4135       \bool_set_true:N \l_@@_stop_loop_bool
4136     }
4137   }
4138   \cs_if_exist:cTF
4139   {
4140     pgf @ sh @ ns @ \@@_env:
4141     - \int_use:N \l_@@_final_i_int
4142     - \int_use:N \l_@@_final_j_int
4143   }
4144   { \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.

```

4145   {

```



```

4146         \cs_set_nopar:cpn
4147         {
4148             @@ _ dotted _
4149             \int_use:N \l_@@_final_i_int -
4150             \int_use:N \l_@@_final_j_int
4151         }
4152         { }
4153     }
4154 }
4155 }
4156 }

```

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

```

4157     \bool_set_false:N \l_@@_stop_loop_bool

```

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

```

4158     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4159     \bool_do_until:Nn \l_@@_stop_loop_bool
4160     {
4161         \int_sub:Nn \l_@@_initial_i_int { #3 }
4162         \int_sub:Nn \l_@@_initial_j_int { #4 }
4163         \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.

```

4164         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4165         \if_int_compare:w #3 = \c_one_int
4166             \bool_set_true:N \l_@@_initial_open_bool
4167         \else:

```

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

```

4168         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4169             \bool_set_true:N \l_@@_initial_open_bool
4170         \fi:
4171     \fi:
4172 \else:
4173     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4174         \if_int_compare:w #4 = \c_one_int
4175             \bool_set_true:N \l_@@_initial_open_bool
4176         \fi:
4177     \else:
4178         \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4179             \if_int_compare:w #4 = -1
4180                 \bool_set_true:N \l_@@_initial_open_bool
4181             \fi:
4182         \fi:
4183     \fi:
4184 \fi:
4185 \bool_if:NTF \l_@@_initial_open_bool
4186 {
4187     \int_add:Nn \l_@@_initial_i_int { #3 }
4188     \int_add:Nn \l_@@_initial_j_int { #4 }
4189     \bool_set_true:N \l_@@_stop_loop_bool
4190 }
4191 {
4192     \cs_if_exist:cTF
4193     {
4194         @@ _ dotted _
4195         \int_use:N \l_@@_initial_i_int -
4196         \int_use:N \l_@@_initial_j_int
4197     }

```

```

4198     {
4199         \int_add:Nn \l_@@_initial_i_int { #3 }
4200         \int_add:Nn \l_@@_initial_j_int { #4 }
4201         \bool_set_true:N \l_@@_initial_open_bool
4202         \bool_set_true:N \l_@@_stop_loop_bool
4203     }
4204     {
4205         \cs_if_exist:cTF
4206         {
4207             pgf @ sh @ ns @ \@@_env:
4208             - \int_use:N \l_@@_initial_i_int
4209             - \int_use:N \l_@@_initial_j_int
4210         }
4211         { \bool_set_true:N \l_@@_stop_loop_bool }
4212     {
4213         \cs_set_nopar:cpn
4214         {
4215             @@ _ dotted _
4216             \int_use:N \l_@@_initial_i_int -
4217             \int_use:N \l_@@_initial_j_int
4218         }
4219         { }
4220     }
4221 }
4222 }
4223 }

```

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.

```

4224 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4225 {
4226     { \int_use:N \l_@@_initial_i_int }

```

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

```

4227     { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4228     { \int_use:N \l_@@_final_i_int }
4229     { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4230     { }
4231 }
4232 }

```

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

```

4233 \cs_new_protected:Npn \@@_open_shorten:
4234 {
4235     \bool_if:NT \l_@@_initial_open_bool
4236     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4237     \bool_if:NT \l_@@_final_open_bool
4238     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4239 }

```

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).

```

4240 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4241 {
4242     \int_set_eq:NN \l_@@_row_min_int \c_one_int

```

```

4243 \int_set_eq:NN \l_@@_col_min_int \c_one_int
4244 \int_set_eq:NN \l_@@_row_max_int \c@iRow
4245 \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`.

```

4246 \seq_if_empty:NF \g_@@_submatrix_seq
4247 {
4248   \seq_map_inline:Nn \g_@@_submatrix_seq
4249   { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4250 }
4251 }

```

#1 and **#2** are the numbers of row and columns of the cell where the command of dotted line (ex.: `\Vdots`) has been issued. **#3**, **#4**, **#5** and **#6** are the specification (in *i* and *j*) of the submatrix we are analyzing.

Here is the programming of that command with the the standard syntax of L3.

```

\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
{
  \bool_if:nT
  {
    \int_compare_p:n { #3 <= #1 <= #5 }
    &&
    \int_compare_p:n { #4 <= #2 <= #6 }
  }
  {
    \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
    \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
    \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
    \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
  }
}

```

However, for efficiency, we will use the following version.

```

4252 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4253 {
4254   \if_int_compare:w #3 > #1
4255   \else:
4256     \if_int_compare:w #1 > #5
4257     \else:
4258       \if_int_compare:w #4 > #2
4259       \else:
4260         \if_int_compare:w #2 > #6
4261         \else:
4262           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4263           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4264           \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4265           \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4266         \fi:
4267       \fi:
4268     \fi:
4269   \fi:
4270 }

4271 \cs_new_protected:Npn \@@_set_initial_coords:
4272 {
4273   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4274   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4275 }
4276 \cs_new_protected:Npn \@@_set_final_coords:
4277 {

```

```

4278 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4279 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4280 }
4281 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4282 {
4283   \pgfpointanchor
4284   {
4285     \@@_env:
4286     - \int_use:N \l_@@_initial_i_int
4287     - \int_use:N \l_@@_initial_j_int
4288   }
4289   { #1 }
4290   \@@_set_initial_coords:
4291 }
4292 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4293 {
4294   \pgfpointanchor
4295   {
4296     \@@_env:
4297     - \int_use:N \l_@@_final_i_int
4298     - \int_use:N \l_@@_final_j_int
4299   }
4300   { #1 }
4301   \@@_set_final_coords:
4302 }
4303 \cs_new_protected:Npn \@@_open_x_initial_dim:
4304 {
4305   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4306   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4307   {
4308     \cs_if_exist:cT
4309     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4310     {
4311       \pgfpointanchor
4312       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4313       { west }
4314       \dim_set:Nn \l_@@_x_initial_dim
4315       { \dim_min:nn { \l_@@_x_initial_dim } { \pgf@x } }
4316     }
4317   }

```

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

```

4318 \dim_compare:nNnT { \l_@@_x_initial_dim } = { \c_max_dim }
4319 {
4320   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4321   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4322   \dim_add:Nn \l_@@_x_initial_dim \col@sep
4323 }
4324 }
4325 \cs_new_protected:Npn \@@_open_x_final_dim:
4326 {
4327   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4328   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4329   {
4330     \cs_if_exist:cT
4331     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4332     {
4333       \pgfpointanchor
4334       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4335       { east }
4336       \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
4337       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4338     }

```

```
4339 }
```

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

```
4340 \dim_compare:nNnT { \l_@@_x_final_dim } = { - \c_max_dim }
4341 {
4342   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4343   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4344   \dim_sub:Nn \l_@@_x_final_dim \col@sep
4345 }
4346 }
```

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.

```
4347 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4348 {
4349   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4350   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4351   {
4352     \@@_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.

```
4353   \group_begin:
4354   \@@_open_shorten:
4355   \int_if_zero:nTF { #1 }
4356   { \color { nicematrix-first-row } }
4357   {
```

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.

```
4358     \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4359     { \color { nicematrix-last-row } }
4360   }
4361   \keys_set:nn { nicematrix / xdots } { #3 }
4362   \@@_color:o \l_@@_xdots_color_tl
4363   \@@_actually_draw_Ldots:
4364   \group_end:
4365 }
4366 }
```

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`.

```
4367 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4368 {
4369   \bool_if:NTF \l_@@_initial_open_bool
4370   {
4371     \@@_open_x_initial_dim:
4372     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4373     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4374   }
4375   { \@@_set_initial_coords_from_anchor:n { base-east } }
```

```

4376 \bool_if:NTF \l_@@_final_open_bool
4377 {
4378   \@@_open_x_final_dim:
4379   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4380   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4381 }
4382 { \@@_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.

```

4383 \bool_lazy_all:nTF
4384 {
4385   \l_@@_initial_open_bool
4386   \l_@@_final_open_bool
4387   { \int_compare_p:nNn { \l_@@_initial_i_int } = { \l_@@_last_row_int } }
4388 }
4389 {
4390   \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4391   \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4392 }

```

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 (?).

```

4393 {
4394   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4395   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4396 }
4397 \@@_draw_line:
4398 }

```

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.

```

4399 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4400 {
4401   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4402   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4403   {
4404     \@@_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.

```

4405 \group_begin:
4406 \@@_open_shorten:
4407 \int_if_zero:nTF { #1 }
4408 { \color { nicematrix-first-row } }
4409 {

```

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.

```

4410 \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4411 { \color { nicematrix-last-row } }
4412 }
4413 \keys_set:nn { nicematrix / xdots } { #3 }
4414 \@@_color:o \l_@@_xdots_color_tl
4415 \@@_actually_draw_Cdots:
4416 \group_end:
4417 }
4418 }

```

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.

```

4419 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4420 {
4421   \bool_if:NTF \l_@@_initial_open_bool
4422     { \@@_open_x_initial_dim: }
4423     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4424   \bool_if:NTF \l_@@_final_open_bool
4425     { \@@_open_x_final_dim: }
4426     { \@@_set_final_coords_from_anchor:n { mid-west } }
4427   \bool_lazy_and:nnTF
4428     { \l_@@_initial_open_bool }
4429     { \l_@@_final_open_bool }
4430   {
4431     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4432     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4433     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4434     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4435     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4436   }
4437   {
4438     \bool_if:NT \l_@@_initial_open_bool
4439       { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4440     \bool_if:NT \l_@@_final_open_bool
4441       { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4442   }
4443   \@@_draw_line:
4444 }

4445 \cs_new_protected:Npn \@@_open_y_initial_dim:
4446 {
4447   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4448   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4449   {
4450     \cs_if_exist:cT
4451       { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4452       {
4453         \pgfpointanchor
4454           { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4455           { north }
4456         \dim_compare:nNnT { \pgf@y } > { \l_@@_y_initial_dim }
4457           { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4458       }
4459   }
4460   \dim_compare:nNnT { \l_@@_y_initial_dim } = { - \c_max_dim }
4461   {
4462     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4463     \dim_set:Nn \l_@@_y_initial_dim
4464       {
4465         \fp_to_dim:n
4466           {
4467             \pgf@y
4468             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4469           }
4470       }
4471   }
4472 }

```

```

4473 \cs_new_protected:Npn \@@_open_y_final_dim:
4474 {
4475   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4476   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4477   {
4478     \cs_if_exist:cT
4479     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4480     {
4481       \pgfpointanchor
4482       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4483       { south }
4484       \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
4485       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4486     }
4487   }
4488   \dim_compare:nNnT { \l_@@_y_final_dim } = { \c_max_dim }
4489   {
4490     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4491     \dim_set:Nn \l_@@_y_final_dim
4492     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4493   }
4494 }

```

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.

```

4495 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4496 {
4497   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4498   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4499   {
4500     \@@_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.

```

4501   \group_begin:
4502     \@@_open_shorten:
4503     \int_if_zero:nTF { #2 }
4504     { \color { nicematrix-first-col } }
4505     {
4506       \int_compare:nNnT { #2 } = { \l_@@_last_col_int }
4507       { \color { nicematrix-last-col } }
4508     }
4509     \keys_set:nn { nicematrix / xdots } { #3 }
4510     \@@_color:o \l_@@_xdots_color_tl
4511     \bool_if:NTF \l_@@_Vbrace_bool
4512     { \@@_actually_draw_Vbrace: }
4513     { \@@_actually_draw_Vdots: }
4514   \group_end:
4515 }
4516 }

```

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`.


```

4517 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4518 {
4519   \bool_lazy_and:nnTF { \l_@@_initial_open_bool } { \l_@@_final_open_bool }
4520   { \@@_actually_draw_Vdots_i: }
4521   { \@@_actually_draw_Vdots_ii: }
4522   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4523   \@@_draw_line:
4524 }

```

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

```

4525 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4526 {
4527   \@@_open_y_initial_dim:
4528   \@@_open_y_final_dim:
4529   \int_if_zero:nTF { \l_@@_initial_j_int }

```

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

```

4530 {
4531   \@@_qpoint:n { col - 1 }
4532   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4533   \dim_sub:Nn \l_@@_x_initial_dim
4534   { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4535 }
4536 {
4537   \bool_lazy_and:nnTF
4538   { \int_compare_p:nNn { \l_@@_last_col_int } > { -2 } }
4539   {
4540     \int_compare_p:nNn
4541     { \l_@@_initial_j_int } = { \g_@@_col_total_int }
4542   }

```

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

```

4543 {
4544   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4545   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4546   \dim_add:Nn \l_@@_x_initial_dim
4547   { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4548 }

```

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

```

4549 {
4550   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4551   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4552   \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4553   \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4554 }
4555 }
4556 }

```

The command `\@@_draw_line:` is in `\@@_actually_draw_Vdots:`

Now, the dotted line is *not* open on both sides (maybe open on only one side).

The main task is to determine the x -value of the dotted line to draw.

The boolean `\l_tmpa_bool` will indicate whether the column is of type 1 or may be considered as if.

```

4557 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4558 {
4559   \bool_set_false:N \l_tmpa_bool
4560   \bool_if:NF \l_@@_initial_open_bool
4561   {
4562     \bool_if:NF \l_@@_final_open_bool
4563     {
4564       \@@_set_initial_coords_from_anchor:n { south-west }
4565       \@@_set_final_coords_from_anchor:n { north-west }
4566       \bool_set:Nn \l_tmpa_bool

```

```

4567         {
4568             \dim_compare_p:nNn
4569             { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4570         }
4571     }
4572 }

```

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

```

4573 \bool_if:NTF \l_@@_initial_open_bool
4574 {
4575     \@@_open_y_initial_dim:
4576     \@@_set_final_coords_from_anchor:n { north }
4577     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4578 }
4579 {
4580     \@@_set_initial_coords_from_anchor:n { south }
4581     \bool_if:NTF \l_@@_final_open_bool
4582     { \@@_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.

```

4583     {
4584         \@@_set_final_coords_from_anchor:n { north }
4585         \dim_compare:nNnF { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4586         {
4587             \dim_set:Nn \l_@@_x_initial_dim
4588             {
4589                 \bool_if:NTF \l_tmpa_bool { \dim_min:nn } { \dim_max:nn }
4590                 \l_@@_x_initial_dim \l_@@_x_final_dim
4591             }
4592         }
4593     }
4594 }
4595 }

```

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`.

```

4596 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4597 {
4598     \bool_if:NTF \l_@@_initial_open_bool
4599     { \@@_open_y_initial_dim: }
4600     { \@@_set_initial_coords_from_anchor:n { south } }
4601     \bool_if:NTF \l_@@_final_open_bool
4602     { \@@_open_y_final_dim: }
4603     { \@@_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.

```

4604 \int_if_zero:nTF { \l_@@_initial_j_int }
4605 {
4606     \@@_qpoint:n { col - 1 }

```

```

4607     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4608     \dim_sub:Nn \l_@@_x_initial_dim
4609     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4610 }

```

Elsewhere, the brace must be drawn left flush.

```

4611 {
4612     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4613     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4614     \dim_add:Nn \l_@@_x_initial_dim
4615     { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4616 }

```

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

```

4617     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4618     \@@_draw_line:
4619 }

```

```

4620 \cs_new:Npn \@@_colsep:
4621 { \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.

```

4622 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4623 {
4624     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4625     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4626     {
4627         \@@_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.

```

4628     \group_begin:
4629     \@@_open_shorten:
4630     \keys_set:nn { nicematrix / xdots } { #3 }
4631     \@@_color:o \l_@@_xdots_color_tl
4632     \@@_actually_draw_Ddots:
4633     \group_end:
4634 }
4635 }

```

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`.

```

4636 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4637 {
4638   \bool_if:NTF \l_@@_initial_open_bool
4639   {
4640     \@@_open_y_initial_dim:
4641     \@@_open_x_initial_dim:
4642   }
4643   { \@@_set_initial_coords_from_anchor:n { south-east } }
4644   \bool_if:NTF \l_@@_final_open_bool
4645   {
4646     \@@_open_x_final_dim:
4647     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4648   }
4649   { \@@_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.

```

4650   \bool_if:NT \l_@@_parallelize_diags_bool
4651   {
4652     \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).

```

4653     \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.

```

4654     {
4655       \dim_gset:Nn \g_@@_delta_x_one_dim
4656       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4657       \dim_gset:Nn \g_@@_delta_y_one_dim
4658       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4659     }

```

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`.

```

4660     {
4661       \dim_compare:nNnF { \g_@@_delta_x_one_dim } = { \c_zero_dim }
4662       {
4663         \dim_set:Nn \l_@@_y_final_dim
4664         {
4665           \l_@@_y_initial_dim +
4666           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4667           \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4668         }
4669       }
4670     }
4671   }
4672   \@@_draw_line:
4673 }

```

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.

```

4674 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4675 {
4676   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4677   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4678   {
4679     \@@_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.

```

4680   \group_begin:

```

```

4681         \@@_open_shorten:
4682         \keys_set:nn { nicematrix / xdots } { #3 }
4683         \@@_color:o \l_@@_xdots_color_tl
4684         \@@_actually_draw_Iddots:
4685     \group_end:
4686 }
4687 }

```

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.`

```

4688 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4689 {
4690     \bool_if:NTF \l_@@_initial_open_bool
4691     {
4692         \@@_open_y_initial_dim:
4693         \@@_open_x_initial_dim:
4694     }
4695     { \@@_set_initial_coords_from_anchor:n { south-west } }
4696     \bool_if:NTF \l_@@_final_open_bool
4697     {
4698         \@@_open_y_final_dim:
4699         \@@_open_x_final_dim:
4700     }
4701     { \@@_set_final_coords_from_anchor:n { north-east } }
4702     \bool_if:NT \l_@@_parallelize_diags_bool
4703     {
4704         \int_gincr:N \g_@@_iddots_int
4705         \int_compare:nNnTF { \g_@@_iddots_int } = { \c_one_int }
4706         {
4707             \dim_gset:Nn \g_@@_delta_x_two_dim
4708             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4709             \dim_gset:Nn \g_@@_delta_y_two_dim
4710             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4711         }
4712         {
4713             \dim_compare:nNnF { \g_@@_delta_x_two_dim } = { \c_zero_dim }
4714             {
4715                 \dim_set:Nn \l_@@_y_final_dim
4716                 {
4717                     \l_@@_y_initial_dim +
4718                     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4719                     \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4720                 }
4721             }
4722         }
4723     }
4724     \@@_draw_line:
4725 }

```

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`

```

4726 \cs_new_protected:Npn \@@_draw_line:
4727 {
4728   \pgfrememberpicturepositiononpagetrue
4729   \pgf@relevantforpicturesizefalse
4730   \bool_lazy_or:nnTF
4731     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4732     { \l_@@_dotted_bool }
4733     { \@@_draw_standard_dotted_line: }
4734     { \@@_draw_unstandard_dotted_line: }
4735 }
```

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.

```

4736 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4737 {
4738   \begin { scope }
4739     \@@_draw_unstandard_dotted_line:o
4740     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4741 }
```

We have used the fact that, in PGF, a 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.

```

4742 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4743 {
4744   \@@_draw_unstandard_dotted_line:nooo
4745   { #1 }
4746   \l_@@_xdots_up_tl
4747   \l_@@_xdots_down_tl
4748   \l_@@_xdots_middle_tl
4749 }
4750 \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.

```

4751 \hook_gput_code:nnn { begindocument } { . }
4752 {
4753   \IfPackageLoadedT { tikz }
4754   {
4755     \tikzset
4756     {
4757       @@_node_above / .style = { sloped , above } ,
4758       @@_node_below / .style = { sloped , below } ,
4759       @@_node_middle / .style =
4760       {
```

```

4761         sloped ,
4762         inner~sep = \c_@@_innersep_middle_dim
4763     }
4764 }
4765 }
4766 }

4767 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4768 {

```

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.

```

4769     \dim_zero_new:N \l_@@_l_dim
4770     \dim_set:Nn \l_@@_l_dim
4771     {
4772         \fp_to_dim:n
4773         {
4774             sqrt
4775             (
4776                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4777                 +
4778                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4779             )
4780         }
4781     }

```

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.

```

4782     \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4783     {
4784         \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4785         \@@_draw_unstandard_dotted_line_i:
4786     }

```

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

```

4787     \bool_if:NT \l_@@_xdots_h_labels_bool
4788     {
4789         \tikzset
4790         {
4791             @@_node_above / .style = { auto = left } ,
4792             @@_node_below / .style = { auto = right } ,
4793             @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4794         }
4795     }
4796     \tl_if_empty:nF { #4 }
4797     { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4798     \draw
4799     [ #1 ]
4800     ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
4801     -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4802     node [ @@_node_below ] { $ \scriptstyle #3 $ }
4803     node [ @@_node_above ] { $ \scriptstyle #2 $ }
4804     ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4805     \end { scope }
4806 }
4807 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }

```

```

4808 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4809 {
4810   \dim_set:Nn \l_tmpa_dim
4811   {
4812     \l_@@_x_initial_dim
4813     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4814     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4815   }
4816   \dim_set:Nn \l_tmpb_dim
4817   {
4818     \l_@@_y_initial_dim
4819     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4820     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4821   }
4822   \dim_set:Nn \l_@@_tmpc_dim
4823   {
4824     \l_@@_x_final_dim
4825     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4826     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4827   }
4828   \dim_set:Nn \l_@@_tmpd_dim
4829   {
4830     \l_@@_y_final_dim
4831     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4832     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4833   }
4834   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4835   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4836   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4837   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4838 }

```

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

```

4839 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4840 {
4841   \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.

```

4842   \dim_zero_new:N \l_@@_l_dim
4843   \dim_set:Nn \l_@@_l_dim
4844   {
4845     \fp_to_dim:n
4846     {
4847       sqrt
4848       (
4849         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4850         +
4851         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4852       )
4853     }
4854   }

```

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.

```

4855   \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4856   {
4857     \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4858     { \@@_draw_standard_dotted_line_i: }
4859   }
4860   \group_end:

```



```

4861 \bool_lazy_all:nF
4862 {
4863   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4864   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4865   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4866 }
4867 { \@@_labels_standard_dotted_line: }
4868 }
4869 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4870 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4871 {

```

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

```

4872 \int_set:Nn \l_tmpa_int
4873 {
4874   \dim_ratio:nn
4875   {
4876     \l_@@_l_dim
4877     - \l_@@_xdots_shorten_start_dim
4878     - \l_@@_xdots_shorten_end_dim
4879   }
4880   { \l_@@_xdots_inter_dim }
4881 }

```

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

```

4882 \dim_set:Nn \l_tmpa_dim
4883 {
4884   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4885   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4886 }
4887 \dim_set:Nn \l_tmpb_dim
4888 {
4889   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4890   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4891 }

```

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.

```

4892 \dim_gadd:Nn \l_@@_x_initial_dim
4893 {
4894   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4895   \dim_ratio:nn
4896   {
4897     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4898     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4899   }
4900   { 2 \l_@@_l_dim }
4901 }
4902 \dim_gadd:Nn \l_@@_y_initial_dim
4903 {
4904   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4905   \dim_ratio:nn
4906   {
4907     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4908     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4909   }
4910   { 2 \l_@@_l_dim }
4911 }
4912 \pgf@relevantforpicturesizefalse
4913 \int_step_inline:nnn { \c_zero_int } { \l_tmpa_int }
4914 {
4915   \pgfpathcircle

```

```

4916         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4917         { \l_@@_xdots_radius_dim }
4918         \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4919         \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4920     }
4921     \pgfusepathqfill
4922 }

4923 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
4924 {
4925     \pgfscope
4926     \pgftransformshift
4927     {
4928         \pgfpointlineattime { 0.5 }
4929         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4930         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4931     }
4932     \fp_set:Nn \l_tmpa_fp
4933     {
4934         atand
4935         (
4936             \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4937             \l_@@_x_final_dim - \l_@@_x_initial_dim
4938         )
4939     }
4940     \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4941     \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4942     \tl_if_empty:NF \l_@@_xdots_middle_tl
4943     {
4944         \begin { pgfscope }
4945         \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4946         \pgfnode
4947         { rectangle }
4948         { center }
4949         {
4950             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4951             {
4952                 $ % $
4953                 \scriptstyle \l_@@_xdots_middle_tl
4954                 $ % $
4955             }
4956         }
4957         { }
4958         {
4959             \pgfsetfillcolor { white }
4960             \pgfusepath { fill }
4961         }
4962         \end { pgfscope }
4963     }
4964     \tl_if_empty:NF \l_@@_xdots_up_tl
4965     {
4966         \pgfnode
4967         { rectangle }
4968         { south }
4969         {
4970             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4971             {
4972                 $ % $
4973                 \scriptstyle \l_@@_xdots_up_tl
4974                 $ % $
4975             }
4976         }
4977         { }

```

```

4978         { \pgfusepath { } }
4979     }
4980     \tl_if_empty:NF \l_@@_xdots_down_tl
4981     {
4982         \pgfnode
4983         { rectangle }
4984         { north }
4985         {
4986             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4987             {
4988                 $ % $
4989                 \scriptstyle \l_@@_xdots_down_tl
4990                 $ % $
4991             }
4992         }
4993         { }
4994         { \pgfusepath { } }
4995     }
4996     \endpgfscope
4997 }

```

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.

```

4998 \hook_gput_code:nnn { begindocument } { . }
4999 {

```

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

```

5000     \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
5001     \cs_new_protected:Npn \@@_Ldots:
5002     { \@@_collect_options:n { \@@_Ldots_i } }
5003     \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5004     {
5005         \int_if_zero:nTF { \c_jCol }
5006         { \@@_error:nn { in~first~col } { \Ldots } }
5007         {
5008             \int_compare:nNnTF { \c_jCol } = { \l_@@_last_col_int }
5009             { \@@_error:nn { in~last~col } { \Ldots } }
5010             {
5011                 \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5012                 { #1 , down = #2 , up = #3 , middle = #4 }
5013             }
5014         }
5015         \bool_if:NF \l_@@_nullify_dots_bool
5016         { \phantom { \ensuremath { \@@_old_ldots: } } }
5017         \bool_gset_true:N \g_@@_empty_cell_bool
5018     }

5019     \cs_new_protected:Npn \@@_Cdots:

```

```

5020 { \@@_collect_options:n { \@@_Cdots_i } }
5021 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5022 {
5023   \int_if_zero:nTF { \c@jCol }
5024   { \@@_error:nn { in~first~col } { \Cdots } }
5025   {
5026     \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5027     { \@@_error:nn { in~last~col } { \Cdots } }
5028     {
5029       \@@_instruction_of_type:nnn { \c_false_bool } { Cdots }
5030       { #1 , down = #2 , up = #3 , middle = #4 }
5031     }
5032   }
5033   \bool_if:NF \l_@@_nullify_dots_bool
5034   { \phantom { \ensuremath { \@@_old_cdots: } } }
5035   \bool_gset_true:N \g_@@_empty_cell_bool
5036 }

5037 \cs_new_protected:Npn \@@_Vdots:
5038 { \@@_collect_options:n { \@@_Vdots_i } }
5039 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5040 {
5041   \int_if_zero:nTF { \c@iRow }
5042   { \@@_error:nn { in~first~row } { \Vdots } }
5043   {
5044     \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
5045     { \@@_error:nn { in~last~row } { \Vdots } }
5046     {
5047       \@@_instruction_of_type:nnn { \c_false_bool } { Vdots }
5048       { #1 , down = #2 , up = #3 , middle = #4 }
5049     }
5050   }
5051   \bool_if:NF \l_@@_nullify_dots_bool
5052   { \phantom { \ensuremath { \@@_old_vdots: } } }
5053   \bool_gset_true:N \g_@@_empty_cell_bool
5054 }

5055 \cs_new_protected:Npn \@@_Ddots:
5056 { \@@_collect_options:n { \@@_Ddots_i } }
5057 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5058 {
5059   \int_case:nnF \c@iRow
5060   {
5061     0 { \@@_error:nn { in~first~row } { \Ddots } }
5062     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5063   }
5064   {
5065     \int_case:nnF \c@jCol
5066     {
5067       0 { \@@_error:nn { in~first~col } { \Ddots } }
5068       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5069     }
5070     {
5071       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5072       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5073       { #1 , down = #2 , up = #3 , middle = #4 }
5074     }
5075   }
5076 }
5077 \bool_if:NF \l_@@_nullify_dots_bool
5078 { \phantom { \ensuremath { \@@_old_ddots: } } }
5079 \bool_gset_true:N \g_@@_empty_cell_bool
5080 }

```

```

5081 \cs_new_protected:Npn \@@_Iddots:
5082   { \@@_collect_options:n { \@@_Iddots_i } }
5083 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5084   {
5085     \int_case:nnF \c@iRow
5086     {
5087       0 { \@@_error:nn { in~first~row } { \Iddots } }
5088       \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5089     }
5090     {
5091       \int_case:nnF \c@jCol
5092       {
5093         0 { \@@_error:nn { in~first~col } { \Iddots } }
5094         \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5095       }
5096       {
5097         \keys_set_known:nn { nicematrix / Ddots } { #1 }
5098         \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { \Iddots }
5099         { #1 , down = #2 , up = #3 , middle = #4 }
5100       }
5101     }
5102     \bool_if:NF \l_@@_nullify_dots_bool
5103     { \phantom { \ensuremath { \@@_old_iddots: } } }
5104     \bool_gset_true:N \g_@@_empty_cell_bool
5105   }
5106 }

```

End of the \AddToHook.

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

```

5107 \keys_define:nn { nicematrix / Ddots }
5108   {
5109     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5110     draw-first .default:n = true ,
5111     draw-first .value_forbidden:n = true
5112   }

```

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

```

5113 \cs_new_protected:Npn \@@_Hspace:
5114   {
5115     \bool_gset_true:N \g_@@_empty_cell_bool
5116     \hspace
5117   }

```

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.

```

5118 \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.

```

5119 \cs_new:Npn \@@_Hdotsfor:
5120   {
5121     \bool_lazy_and:nnTF
5122     { \int_if_zero_p:n { \c@jCol } }
5123     { \int_if_zero_p:n { \l_@@_first_col_int } }
5124     {
5125       \bool_if:NTF \g_@@_after_col_zero_bool
5126       {
5127         \multicolumn { 1 } { c } { }
5128         \@@_Hdotsfor_i:

```

```

5129     }
5130     { \@@_fatal:n { Hdotsfor~in~col-0 } }
5131 }
5132 {
5133     \multicolumn { 1 } { c } { }
5134     \@@_Hdotsfor_i:
5135 }
5136 }

```

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:`).

```

5137 \hook_gput_code:nnn { begindocument } { . }
5138 {

```

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

```

5139     \cs_new_protected:Npn \@@_Hdotsfor_i:
5140     { \@@_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`).

```

5141     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5142     \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5143     {
5144         \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5145         {
5146             \@@_Hdotsfor:nnnn
5147             { \int_use:N \c@iRow }
5148             { \int_use:N \c@jCol }
5149             { #2 }
5150             {
5151                 #1 , #3 ,
5152                 down = \exp_not:n { #4 } ,
5153                 up = \exp_not:n { #5 } ,
5154                 middle = \exp_not:n { #6 }
5155             }
5156         }
5157         \prg_replicate:nn { #2 - 1 }
5158         {
5159             &
5160             \multicolumn { 1 } { c } { }
5161             \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5162         }
5163     }
5164 }

```

```

5165 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5166 {
5167     \bool_set_false:N \l_@@_initial_open_bool
5168     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5169     \int_set:Nn \l_@@_initial_i_int { #1 }
5170     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5171     \int_compare:nNnTF { #2 } = { \c_one_int }
5172     {
5173         \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5174         \bool_set_true:N \l_@@_initial_open_bool
5175     }
5176     {
5177         \cs_if_exist:cTF
5178         {

```

```

5179         pgf @ sh @ ns @ \@@_env:
5180         - \int_use:N \l_@@_initial_i_int
5181         - \int_eval:n { #2 - 1 }
5182     }
5183     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5184     {
5185         \int_set:Nn \l_@@_initial_j_int { #2 }
5186         \bool_set_true:N \l_@@_initial_open_bool
5187     }
5188 }
5189 \int_compare:nNnTF { #2 + #3 - 1 } = { \c@jCol }
5190 {
5191     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5192     \bool_set_true:N \l_@@_final_open_bool
5193 }
5194 {
5195     \cs_if_exist:cTF
5196     {
5197         pgf @ sh @ ns @ \@@_env:
5198         - \int_use:N \l_@@_final_i_int
5199         - \int_eval:n { #2 + #3 }
5200     }
5201     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5202     {
5203         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5204         \bool_set_true:N \l_@@_final_open_bool
5205     }
5206 }
5207 \group_begin:
5208 \@@_open_shorten:
5209 \int_if_zero:nTF { #1 }
5210 { \color { nicematrix-first-row } }
5211 {
5212     \int_compare:nNnT { #1 } = { \g_@@_row_total_int }
5213     { \color { nicematrix-last-row } }
5214 }
5215 \keys_set:nn { nicematrix / xdots } { #4 }
5216 \@@_color:o \l_@@_xdots_color_tl
5217 \@@_actually_draw_Ldots:
5218 \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).

```

5219     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5220     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5221 }

```

```

5222 \hook_gput_code:nnn { begindocument } { . }
5223 {
5224     \cs_new_protected:Npn \@@_Vdotsfor:
5225     { \@@_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`).

```

5226     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5227     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5228     {
5229         \bool_gset_true:N \g_@@_empty_cell_bool
5230         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5231         {
5232             \@@_Vdotsfor:nnnn

```

```

5233         { \int_use:N \c@iRow }
5234         { \int_use:N \c@jCol }
5235         { #2 }
5236         {
5237             #1 , #3 ,
5238             down = \exp_not:n { #4 } ,
5239             up = \exp_not:n { #5 } ,
5240             middle = \exp_not:n { #6 }
5241         }
5242     }
5243 }
5244 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5245 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5246 {
5247     \bool_set_false:N \l_@@_initial_open_bool
5248     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5249     \int_set:Nn \l_@@_initial_j_int { #2 }
5250     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5251     \int_compare:nNnTF { #1 } = { \c_one_int }
5252     {
5253         \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5254         \bool_set_true:N \l_@@_initial_open_bool
5255     }
5256     {
5257         \cs_if_exist:cTF
5258         {
5259             pgf @ sh @ ns @ \@@_env:
5260             - \int_eval:n { #1 - 1 }
5261             - \int_use:N \l_@@_initial_j_int
5262         }
5263         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5264         {
5265             \int_set:Nn \l_@@_initial_i_int { #1 }
5266             \bool_set_true:N \l_@@_initial_open_bool
5267         }
5268     }
5269     \int_compare:nNnTF { #1 + #3 - 1 } = { \c@iRow }
5270     {
5271         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5272         \bool_set_true:N \l_@@_final_open_bool
5273     }
5274     {
5275         \cs_if_exist:cTF
5276         {
5277             pgf @ sh @ ns @ \@@_env:
5278             - \int_eval:n { #1 + #3 }
5279             - \int_use:N \l_@@_final_j_int
5280         }
5281         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5282         {
5283             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5284             \bool_set_true:N \l_@@_final_open_bool
5285         }
5286     }

```



```

5287 \group_begin:
5288 \@@_open_shorten:
5289 \int_if_zero:nTF { #2 }
5290 { \color { nicematrix-first-col } }
5291 {
5292   \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
5293   { \color { nicematrix-last-col } }
5294 }
5295 \keys_set:nn { nicematrix / xdots } { #4 }
5296 \@@_color:o \l_@@_xdots_color_tl
5297 \bool_if:NTF \l_@@_Vbrace_bool
5298 { \@@_actually_draw_Vbrace: }
5299 { \@@_actually_draw_Vdots: }
5300 \group_end:

```

We declare all the cells concerned by the `\Vdotsfor` as “dotted” (for the dotted lines created by `\Cdots`, `\Ldots`, etc., this job is done by `\@@_find_extremities_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5301 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5302 { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5303 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5304 \NewDocumentCommand \@@_rotate: { 0 { } }
5305 {
5306   \bool_gset_true:N \g_@@_rotate_bool
5307   \keys_set:nn { nicematrix / rotate } { #1 }
5308   \ignorespaces
5309 }

5310 \keys_define:nn { nicematrix / rotate }
5311 {
5312   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5313   c .value_forbidden:n = true ,
5314   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5315 }

```

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

```

5316 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop

```

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

```

5317 {
5318   \tl_if_empty:nTF { #2 }
5319     { #1 }
5320     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5321 }
5322 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5323 { \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).

```

5324 \hook_gput_code:nnn { begindocument } { . }
5325 {

```

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

```

5326   \tl_set_rescan:Nnn \l_tmpa_tl { }
5327   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5328   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5329     {
5330       \group_begin:
5331       \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5332       \@@_color:o \l_@@_xdots_color_tl
5333       \use:e
5334       {
5335         \@@_line_i:nn
5336         { \@@_double_int_eval:n #2 - \q_stop }
5337         { \@@_double_int_eval:n #3 - \q_stop }
5338       }
5339       \group_end:
5340     }
5341 }
5342 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5343 {
5344   \bool_set_false:N \l_@@_initial_open_bool
5345   \bool_set_false:N \l_@@_final_open_bool
5346   \bool_lazy_or:nnTF
5347     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5348     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5349     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

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

```

5350   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5351 }
5352 \hook_gput_code:nnn { begindocument } { . }
5353 {
5354   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5355   {

```

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:`.

```

5356     \c_@@_pgfortikzpicture_tl
5357     \@@_draw_line_iii:nn { #1 } { #2 }
5358     \c_@@_endpgfortikzpicture_tl
5359   }
5360 }

```

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

```

5361 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5362 {
5363   \pgfrememberpicturepositiononpagetrue

```

```

5364 \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5365 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5366 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5367 \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5368 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5369 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5370 \@@_draw_line:
5371 }

```

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.

```

5372 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNnT { \c@iRow } < }
5373 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNnF { \c@jCol } < }

```

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

```

5374 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5375 {
5376   \tl_gput_right:Ne \g_@@_row_style_tl
5377   {

```

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

```

5378 \exp_not:N
5379 \@@_if_row_less_than:nn
5380 { \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`).

```

5381 {
5382   \exp_not:N
5383   \@@_if_col_greater_than:nn
5384   { \int_eval:n { \c@jCol } }
5385   { \exp_not:n { #1 } \scan_stop: }
5386 }
5387 }
5388 }
5389 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

```

```

5390 \keys_define:nn { nicematrix / RowStyle }
5391 {
5392   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5393   cell-space-top-limit .value_required:n = true ,
5394   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5395   cell-space-bottom-limit .value_required:n = true ,

```

```

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

```

```

5423 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5424 {
5425     \group_begin:
5426     \tl_clear:N \l_@@_fill_tl
5427     \tl_clear:N \l_@@_opacity_tl
5428     \tl_clear:N \l_@@_color_tl
5429     \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5430     \dim_zero:N \l_@@_rounded_corners_dim
5431     \dim_zero:N \l_tmpa_dim
5432     \dim_zero:N \l_tmpb_dim
5433     \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5434     \tl_if_empty:NF \l_@@_fill_tl
5435     {
5436         \@@_add_opacity_to_fill:
5437         \tl_gput_right:Ne \g_@@_pre_code_before_tl
5438         {

```

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

```

5439         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5440         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5441         {
5442             \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5443             - *
5444         }
5445         { \dim_use:N \l_@@_rounded_corners_dim }
5446     }
5447 }
5448 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5449     \dim_compare:nNnT { \l_tmpa_dim } > { \c_zero_dim }
5450     {
5451         \@@_put_in_row_style:e
5452         {

```

```

5453         \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5454         {

```

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

```

5455         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5456         { \dim_use:N \l_tmpa_dim }
5457     }
5458 }
5459 }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

5460 \dim_compare:nNnT { \l_tmpb_dim } > { \c_zero_dim }
5461 {
5462     \@@_put_in_row_style:e
5463     {
5464         \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5465         {
5466             \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5467             { \dim_use:N \l_tmpb_dim }
5468         }
5469     }
5470 }

```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

5471 \tl_if_empty:NF \l_@@_color_tl
5472 {
5473     \@@_put_in_row_style:e
5474     {
5475         \mode_leave_vertical:
5476         \@@_color:n { \l_@@_color_tl }
5477     }
5478 }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5479 \bool_if:NT \l_@@_bold_row_style_bool
5480 {
5481     \@@_put_in_row_style:n
5482     {
5483         \exp_not:n
5484         {
5485             \if_mode_math:
5486             $ % $
5487             \bfseries \boldmath
5488             $ % $
5489             \else:
5490             \bfseries \boldmath
5491             \fi:
5492         }
5493     }
5494 }
5495 \group_end:
5496 \g_@@_row_style_tl
5497 \ignorespaces
5498 }

```

The following commande must *not* be protected.

```

5499 \cs_new:Npn \@@_rounded_from_row:n #1
5500 {
5501     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

In the following code, the “`- 1`” is *not* a subtraction.

```

5502     { \int_eval:n { #1 } - 1 }
5503     {
5504         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5505         - \exp_not:n { \int_use:N \c@jCol }

```

```

5506     }
5507     { \dim_use:N \l_@@_rounded_corners_dim }
5508 }

```

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).

```

5509 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5510 {

```

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.

```

5511     \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`.

```

5512     \str_if_in:nnF { #1 } { !! }
5513     {
5514         \seq_map_indexed_inline:Nn \g_@@_colors_seq

```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```

5515         { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5516     }
5517     \int_if_zero:nTF { \l_tmpa_int }

```

First, the case where the color is a *new* color (not in the sequence).

```

5518     {
5519         \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5520         \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5521     }

```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```

5522     { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5523 }
5524 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5525 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }

```

The following command must be used within a `\pgfpicture`.

```

5526 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5527 {
5528   \dim_compare:nNtT { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim }
5529   {

```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5530   \group_begin:
5531   \pgfsetcornersarced
5532   {
5533     \pgfpoint
5534     { \l_@@_tab_rounded_corners_dim }
5535     { \l_@@_tab_rounded_corners_dim }
5536   }

```

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.

```

5537   \bool_if:NTF \l_@@_hvlines_bool
5538   {
5539     \pgfpathrectanglecorners
5540     {
5541       \pgfpointadd
5542       { \@@_qpoint:n { row-1 } }
5543       { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5544     }
5545     {
5546       \pgfpointadd
5547       {
5548         \@@_qpoint:n
5549         { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5550       }
5551       { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5552     }
5553   }
5554   {
5555     \pgfpathrectanglecorners
5556     { \@@_qpoint:n { row-1 } }
5557     {
5558       \pgfpointadd
5559       {
5560         \@@_qpoint:n
5561         { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5562       }
5563       { \pgfpoint \c_zero_dim \arrayrulewidth }
5564     }
5565   }
5566   \pgfusepath { clip }
5567   \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5568   }
5569 }

```

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-t1`).

```

5570 \cs_new_protected:Npn \@@_actually_color:
5571 {
5572   \pgfpicture
5573   \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.

```

5574 \@@_clip_with_rounded_corners:
5575 \seq_map_indexed_inline:Nn \g_@@_colors_seq
5576 {
5577   \int_compare:nNnTF { ##1 } = { \c_one_int }
5578   {
5579     \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5580     \use:c { g_@@_color _ 1 _tl }
5581     \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5582   }
5583   {
5584     \begin { pgfscope }
5585       \@@_color_opacity: ##2
5586       \use:c { g_@@_color _ ##1 _tl }
5587       \tl_gclear:c { g_@@_color _ ##1 _tl }
5588       \pgfusepath { fill }
5589     \end { pgfscope }
5590   }
5591 }
5592 \endpgfpicture
5593 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5594 \cs_new_protected:Npn \@@_color_opacity:
5595 {
5596   \peek_meaning:NTF [
5597     { \@@_color_opacity:w }
5598     { \@@_color_opacity:w [ ] }
5599   }

```

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.

```

5600 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5601 {
5602   \tl_clear:N \l_tmpa_tl
5603   \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.

```

5604   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5605   \tl_if_empty:NTF \l_tmpb_tl
5606     { \@declaredcolor }
5607     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5608 }

```

The following set of keys is used by the command `\@@_color_opacity:w`.

```

5609 \keys_define:nn { nicematrix / color-opacity }
5610 {
5611   opacity .tl_set:N          = \l_tmpa_tl ,
5612   opacity .value_required:n = true
5613 }

```

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

```

5614 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5615 {
5616   \def \l_@@_rows_tl { #1 }
5617   \def \l_@@_cols_tl { #2 }
5618   \@@_cartesian_path:
5619 }

```


Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```
5620 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5621 {
5622   \tl_if_blank:nF { #2 }
5623   {
5624     \@@_add_to_colors_seq:en
5625     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5626     { \@@_cartesian_color:nn { #3 } { - } }
5627   }
5628 }
```

Here an example: `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```
5629 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5630 {
5631   \tl_if_blank:nF { #2 }
5632   {
5633     \@@_add_to_colors_seq:en
5634     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5635     { \@@_cartesian_color:nn { - } { #3 } }
5636   }
5637 }
```

Here is an example: `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```
5638 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5639 {
5640   \tl_if_blank:nF { #2 }
5641   {
5642     \@@_add_to_colors_seq:en
5643     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5644     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5645   }
5646 }
```

The last argument is the radius of the corners of the rectangle.

```
5647 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5648 {
5649   \tl_if_blank:nF { #2 }
5650   {
5651     \@@_add_to_colors_seq:en
5652     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5653     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5654   }
5655 }
```

The last argument is the radius of the corners of the rectangle.

```
5656 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5657 {
5658   \@@_cut_on_hyphen:w #1 \q_stop
5659   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5660   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5661   \@@_cut_on_hyphen:w #2 \q_stop
5662   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5663   \tl_set:Ne \l_@@_cols_tl { \l_@@_tmpd_tl - \l_tmpb_tl }
```

The command `\@@_cartesian_path:n` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```
5664 \@@_cartesian_path:n { #3 }
5665 }
```

Here is an example: `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5666 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5667 {
5668   \clist_map_inline:nn { #3 }
5669     { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5670 }

5671 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5672 {
5673   \int_step_inline:nn { \c@iRow }
5674   {
5675     \int_step_inline:nn { \c@jCol }
5676     {
5677       \int_if_even:nTF { #####1 + ##1 }
5678         { \@@_cellcolor [ #1 ] { #2 } }
5679         { \@@_cellcolor [ #1 ] { #3 } }
5680       { ##1 - #####1 }
5681     }
5682   }
5683 }

```

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”.

```

5684 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5685 {
5686   \@@_rectanglecolor [ #1 ] { #2 }
5687   { 1 - 1 }
5688   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5689 }

5690 \keys_define:nn { nicematrix / rowcolors }
5691 {
5692   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5693   respect-blocks .default:n = true ,
5694   cols .tl_set:N = \l_@@_cols_tl ,
5695   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5696   restart .default:n = true ,
5697   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5698 }

```

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*.

```

5699 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5700 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5701   \group_begin:
5702   \seq_clear_new:N \l_@@_colors_seq
5703   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5704   \tl_clear_new:N \l_@@_cols_tl
5705   \tl_set:Nn \l_@@_cols_tl { - }
5706   \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).

```

5707 \int_zero_new:N \l_@@_color_int
5708 \int_set_eq:NN \l_@@_color_int \c_one_int
5709 \bool_if:NT \l_@@_respect_blocks_bool
5710 {

```

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`).

```

5711 \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5712 \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5713 { \@@_not_in_exterior_p:nnnnn ##1 }
5714 }
5715 \pgfpicture
5716 \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5717 \clist_map_inline:nn { #2 }
5718 {
5719 \tl_set:Nn \l_tmpa_tl { ##1 }
5720 \tl_if_in:NnTF \l_tmpa_tl { - }
5721 { \@@_cut_on_hyphen:w ##1 \q_stop }
5722 { \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.

```

5723 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5724 \int_set:Nn \l_@@_color_int
5725 { \bool_if:NNTF \l_@@_rowcolors_restart_bool { 1 } { \l_tmpa_tl } }
5726 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5727 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5728 {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5729 \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).

```

5730 \bool_if:NT \l_@@_respect_blocks_bool
5731 {
5732 \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5733 { \@@_intersect_our_row_p:nnnnn ###1 }
5734 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5735 }
5736 \tl_set:Ne \l_@@_rows_tl
5737 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5738 \tl_set:Ne \l_@@_color_tl
5739 {
5740 \@@_color_index:n
5741 {
5742 \int_mod:nn
5743 { \l_@@_color_int - 1 }
5744 { \seq_count:N \l_@@_colors_seq }
5745 + 1
5746 }
5747 }
5748 \tl_if_empty:NF \l_@@_color_tl
5749 {
5750 \@@_add_to_colors_seq:ee
5751 { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5752 { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5753 }
5754 \int_incr:N \l_@@_color_int

```

```

5755         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5756     }
5757 }
5758 \endpgfpicture
5759 \group_end:
5760 }

```

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.

```

5761 \cs_new:Npn \@@_color_index:n #1
5762 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5763     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5764     { \@@_color_index:n { #1 - 1 } }
5765     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5766 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by curryfication.

```

5767 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5768 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5769 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5770 {
5771     \int_compare:nNnT { #3 } > { \l_tmpb_int }
5772     { \int_set:Nn \l_tmpb_int { #3 } }
5773 }

```

```

5774 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5775 {
5776     \int_if_zero:nTF { #4 }
5777     { \prg_return_false: }
5778     {
5779         \int_compare:nNnTF { #2 } > { \c@jCol }
5780         { \prg_return_false: }
5781         { \prg_return_true: }
5782     }
5783 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5784 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5785 {
5786     \int_compare:nNnTF { #1 } > { \l_tmpa_int }
5787     { \prg_return_false: }
5788     {
5789         \int_compare:nNnTF { \l_tmpa_int } > { #3 }
5790         { \prg_return_false: }
5791         { \prg_return_true: }
5792     }
5793 }

```

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`).

```

5794 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5795 {
5796   \dim_compare:nNnTF { #1 } = { \c_zero_dim }
5797   {
5798     \bool_if:NTF \l_@@_nocolor_used_bool
5799     { \@@_cartesian_path_normal_ii: }
5800     {
5801       \clist_if_empty:NTF \l_@@_corners_cells_clist
5802       { \@@_cartesian_path_normal_i:n { #1 } }
5803       { \@@_cartesian_path_normal_ii: }
5804     }
5805   }
5806   { \@@_cartesian_path_normal_i:n { #1 } }
5807 }

```

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.

```

5808 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5809 {
5810   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5811   \clist_map_inline:Nn \l_@@_cols_tl
5812   {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

5813     \def \l_tmpa_tl { ##1 }
5814     \tl_if_in:NnTF \l_tmpa_tl { - }
5815     { \@@_cut_on_hyphen:w ##1 \q_stop }
5816     { \def \l_tmpb_tl { ##1 } } % 2025-04-16
5817     \tl_if_empty:NTF \l_tmpa_tl
5818     { \def \l_tmpa_tl { 1 } }
5819     {
5820       \str_if_eq:eeT { \l_tmpa_tl } { * }
5821       { \def \l_tmpa_tl { 1 } }
5822     }
5823     \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_col_total_int }
5824     { \@@_error:n { Invalid~col~number } }
5825     \tl_if_empty:NTF \l_tmpb_tl
5826     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5827     {
5828       \str_if_eq:eeT { \l_tmpb_tl } { * }
5829       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5830     }
5831     \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_col_total_int }
5832     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5833     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5834     \@@_qpoint:n { col - \l_tmpa_tl }
5835     \int_compare:nNnTF { \l_@@_first_col_int } = { \l_tmpa_tl }
5836     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5837     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5838     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5839     \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.

```

5840     \clist_map_inline:Nn \l_@@_rows_tl
5841     {
5842       \def \l_tmpa_tl { ####1 }
5843       \tl_if_in:NnTF \l_tmpa_tl { - }
5844       { \@@_cut_on_hyphen:w ####1 \q_stop }

```

```

5845     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5846 \tl_if_empty:NTF \l_tmpa_tl
5847   { \def \l_tmpa_tl { 1 } }
5848   {
5849     \str_if_eq:eeT { \l_tmpa_tl } { * }
5850     { \def \l_tmpa_tl { 1 } }
5851   }
5852 \tl_if_empty:NTF \l_tmpb_tl
5853   { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5854   {
5855     \str_if_eq:eeT { \l_tmpb_tl } { * }
5856     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5857   }
5858 \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_row_total_int }
5859   { \@@_error:n { Invalid~row~number } }
5860 \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_row_total_int }
5861   { \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`.

```

5862 \cs_if_exist:cF
5863   { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5864   {
5865     \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5866     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5867     \@@_qpoint:n { row - \l_tmpa_tl }
5868     \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5869     \pgfpathrectanglecorners
5870       { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5871       { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5872   }
5873 }
5874 }
5875 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5876 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5877   {
5878     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5879     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5880 \clist_map_inline:Nn \l_@@_cols_tl
5881   {
5882     \@@_qpoint:n { col - ##1 }
5883     \int_compare:nNnTF { \l_@@_first_col_int } = { ##1 }
5884       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5885       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5886     \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5887     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5888 \clist_map_inline:Nn \l_@@_rows_tl
5889   {
5890     \@@_if_in_corner:nF { #####1 - ##1 }
5891     {
5892       \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5893       \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5894       \@@_qpoint:n { row - #####1 }
5895       \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5896       \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5897       {
5898         \pgfpathrectanglecorners
5899           { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5900           { \pgfpoint \l_tmpa_dim \l_tmpb_dim }

```

```

5901     }
5902   }
5903 }
5904 }
5905 }

```

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`).

```

5906 \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

```

5907 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5908 {
5909   \bool_set_true:N \l_@@_nocolor_used_bool
5910   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5911   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5912   \clist_map_inline:Nn \l_@@_rows_tl
5913   {
5914     \clist_map_inline:Nn \l_@@_cols_tl
5915     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - ####1 } { } }
5916   }
5917 }

```

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`.

```

5918 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5919 {
5920   \clist_set_eq:NN \l_tmpa_clist #1
5921   \clist_clear:N #1
5922   \clist_map_inline:Nn \l_tmpa_clist
5923   {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

5924     \def \l_tmpa_tl { ##1 }
5925     \tl_if_in:NnTF \l_tmpa_tl { - }
5926     { \@@_cut_on_hyphen:w ##1 \q_stop }
5927     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5928     \bool_lazy_or:nnT
5929     { \str_if_eq_p:ee { \l_tmpa_tl } { * } }
5930     { \tl_if_blank_p:o \l_tmpa_tl }
5931     { \def \l_tmpa_tl { 1 } }
5932     \bool_lazy_or:nnT
5933     { \str_if_eq_p:ee { \l_tmpb_tl } { * } }
5934     { \tl_if_blank_p:o \l_tmpb_tl }
5935     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5936     \int_compare:nNnT { \l_tmpb_tl } > { #2 }
5937     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5938     \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
5939     { \clist_put_right:Nn #1 { ####1 } }
5940   }
5941 }

```

The following command will be linked to `\cellcolor` in the tabular.

```

5942 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5943 {
5944   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5945   {

```

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).

```

5946      \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5947      { \int_use:N \c@iRow - \int_use:N \c@jCol }
5948    }
5949    \ignorespaces
5950  }

```

The following command will be linked to \rowcolor in the tabular.

```

5951 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5952 {
5953   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5954   {
5955     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5956     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5957     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5958   }
5959   \ignorespaces
5960 }

```

The following command will be linked to \rowcolors in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5961 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5962 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around #2 and #3 are mandatory.

The following command will be linked to \rowlistcolors in the tabular.

```

5963 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5964 {

```

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.

```

5965   \seq_gclear:N \g_tmpa_seq
5966   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5967   { \@@_rowlistcolors_tabular:nnnn ##1 }
5968   \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.

```

5969   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5970   {
5971     { \int_use:N \c@iRow }
5972     { \exp_not:n { #1 } }
5973     { \exp_not:n { #2 } }
5974     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5975   }
5976   \ignorespaces
5977 }

```

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).


```

5978 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
5979 {
5980   \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).

```

5981   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5982   {
5983     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5984     {
5985       \@@_rowlistcolors
5986       [ \exp_not:n { #2 } ]
5987       { #1 - \int_eval:n { \c@iRow - 1 } }
5988       { \exp_not:n { #3 } }
5989       [ \exp_not:n { #4 } ]
5990     }
5991   }
5992 }

```

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.

```

5993 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5994 {
5995   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5996   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5997   \seq_gclear:N \g_@@_rowlistcolors_seq
5998 }

```

```

5999 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6000 {
6001   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6002   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6003 }

```

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.

```

6004 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
6005 {

```

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).

```

6006   \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
6007   {

```

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).

```

6008     \tl_gput_left:Ne \g_@@_pre_code_before_tl
6009     {
6010       \exp_not:N \columncolor [ #1 ]
6011       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6012     }
6013   }
6014 }

```

```

6015 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6016 {
6017   \clist_map_inline:nn { #1 }

```

```

6018     {
6019         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6020         { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6021         \columncolor { nocolor } { ##1 }
6022     }
6023 }
6024 \cs_new_protected:Npn \@@_EmptyRow:n #1
6025 {
6026     \clist_map_inline:nn { #1 }
6027     {
6028         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6029         { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6030         \rowcolor { nocolor } { ##1 }
6031     }
6032 }

```

22 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnstype` 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`.

```

6033 \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`.

```

6034 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6035 {
6036     \int_if_zero:nTF { \l_@@_first_col_int }
6037     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6038     {
6039         \int_if_zero:nTF { \c@jCol }
6040         {
6041             \int_compare:nNnF { \c@iRow } = { -1 }
6042             {
6043                 \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int - 1 }
6044                 { #1 }
6045             }
6046         }
6047         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6048     }
6049 }

```

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.

```

6050 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6051 {
6052     \int_if_zero:nF { \c@iRow }
6053     {

```

```

6054 \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
6055 {
6056 \int_compare:nNnT { \c@jCol } > { \c_zero_int }
6057 { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6058 }
6059 }
6060 }

```

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`.

```

6061 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6062 {
6063 \IfPackageLoadedTF { tikz }
6064 {
6065 \IfPackageLoadedTF { booktabs }
6066 { #2 }
6067 { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6068 }
6069 { \@@_error:nn { TopRule~without~tikz } { #1 } }
6070 }
6071 \NewExpandableDocumentCommand { \@@_TopRule } { }
6072 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
6073 \cs_new:Npn \@@_TopRule_i:
6074 {
6075 \noalign \bgroup
6076 \peek_meaning:NTF [
6077 { \@@_TopRule_ii: }
6078 { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6079 }
6080 \NewDocumentCommand \@@_TopRule_ii: { o }
6081 {
6082 \tl_gput_right:Ne \g_@@_pre_code_after_tl
6083 {
6084 \@@_hline:n
6085 {
6086 position = \int_eval:n { \c@iRow + 1 } ,
6087 tikz =
6088 {
6089 line~width = #1 ,
6090 yshift = 0.25 \arrayrulewidth ,
6091 shorten~< = - 0.5 \arrayrulewidth
6092 } ,
6093 total~width = #1
6094 }
6095 }
6096 \skip_vertical:n { \belowrulesep + #1 }
6097 \egroup
6098 }
6099 \NewExpandableDocumentCommand { \@@_BottomRule } { }
6100 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }
6101 \cs_new:Npn \@@_BottomRule_i:
6102 {
6103 \noalign \bgroup
6104 \peek_meaning:NTF [
6105 { \@@_BottomRule_ii: }
6106 { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6107 }
6108 \NewDocumentCommand \@@_BottomRule_ii: { o }
6109 {

```

```

6110 \tl_gput_right:Ne \g_@@_pre_code_after_tl
6111 {
6112   \@@_hline:n
6113   {
6114     position = \int_eval:n { \c@iRow + 1 } ,
6115     tikz =
6116     {
6117       line-width = #1 ,
6118       yshift = 0.25 \arrayrulewidth ,
6119       shorten~< = - 0.5 \arrayrulewidth
6120     } ,
6121     total-width = #1 ,
6122   }
6123 }
6124 \skip_vertical:N \aboverulesep
6125 \@@_create_row_node_i:
6126 \skip_vertical:n { #1 }
6127 \egroup
6128 }
6129 \NewExpandableDocumentCommand { \@@_MidRule } { } { }
6130 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6131 \cs_new:Npn \@@_MidRule_i:
6132 {
6133   \noalign \bgroup
6134   \peek_meaning:NTF [
6135     { \@@_MidRule_ii: }
6136     { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6137   }
6138 \NewDocumentCommand \@@_MidRule_ii: { o }
6139 {
6140   \skip_vertical:N \aboverulesep
6141   \@@_create_row_node_i:
6142   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6143   {
6144     \@@_hline:n
6145     {
6146       position = \int_eval:n { \c@iRow + 1 } ,
6147       tikz =
6148       {
6149         line-width = #1 ,
6150         yshift = 0.25 \arrayrulewidth ,
6151         shorten~< = - 0.5 \arrayrulewidth
6152       } ,
6153       total-width = #1 ,
6154     }
6155   }
6156   \skip_vertical:n { \belowrulesep + #1 }
6157   \egroup
6158 }

```

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.

```

6159 \keys_define:nn { nicematrix / Rules }
6160 {
6161   position .int_set:N = \l_@@_position_int ,
6162   position .value_required:n = true ,

```

```

6163     start .int_set:N = \l_@@_start_int ,
6164     end .code:n =
6165         \bool_lazy_or:nnTF
6166         { \tl_if_empty_p:n { #1 } }
6167         { \str_if_eq_p:ee { #1 } { last } }
6168         { \int_set_eq:NN \l_@@_end_int \c@jCol }
6169         { \int_set:Nn \l_@@_end_int { #1 } }
6170     }

```

It's possible that the rule won't be drawn continuously from `start` to `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That's why an analysis is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```

6171 \keys_define:nn { nicematrix / RulesBis }
6172 {
6173     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6174     multiplicity .initial:n = 1 ,
6175     dotted .bool_set:N = \l_@@_dotted_bool ,
6176     dotted .initial:n = false ,
6177     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]`.

```

6178     color .code:n =
6179         \@@_set_CTarc:n { #1 }
6180         \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6181     color .value_required:n = true ,
6182     sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6183     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.

```

6184     tikz .code:n =
6185         \IfPackageLoadedTF { tikz }
6186         { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6187         { \@@_error:n { tikz~without~tikz } } ,
6188     tikz .value_required:n = true ,
6189     total-width .dim_set:N = \l_@@_rule_width_dim ,
6190     total-width .value_required:n = true ,
6191     width .meta:n = { total-width = #1 } ,
6192     unknown .code:n =
6193         \@@_unknown_key:nn
6194         { nicematrix / RulesBis }
6195         { Unknown~key~for~RulesBis }
6196     }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6197 \cs_new_protected:Npn \@@_vline:n #1
6198 {

```

The group is for the options.

```

6199     \group_begin:
6200     \int_set_eq:NN \l_@@_end_int \c@iRow
6201     \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).

```

6202 \int_compare:nNt { \l_@@_position_int } < { \c@jCol + 2 }
6203 \@@_vline_i:
6204 \group_end:
6205 }

6206 \cs_new_protected:Npn \@@_vline_i:
6207 {

```

`\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`.

```

6208 \tl_set:N \l_tmpb_tl { \int_use:N \l_@@_position_int }
6209 \int_step_variable:nnN \l_@@_start_int \l_@@_end_int
6210 \l_tmpa_tl
6211 {

```

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.

```

6212 \bool_gset_true:N \g_tmpa_bool

6213 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6214 { \@@_test_vline_in_block:nnnnn ##1 }
6215 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6216 { \@@_test_vline_in_block:nnnnn ##1 }
6217 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6218 { \@@_test_vline_in_stroken_block:nnnn ##1 }
6219 \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_v: }
6220 \bool_if:NTF \g_tmpa_bool
6221 {
6222 \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.

```

6223 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6224 }
6225 {
6226 \int_compare:nNt { \l_@@_local_start_int } > { \c_zero_int }
6227 {
6228 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6229 \@@_vline_ii:
6230 \int_zero:N \l_@@_local_start_int
6231 }
6232 }
6233 }
6234 \int_compare:nNt { \l_@@_local_start_int } > { \c_zero_int }
6235 {
6236 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6237 \@@_vline_ii:
6238 }
6239 }

```

```

6240 \cs_new_protected:Npn \@@_test_in_corner_v:
6241 {
6242 \int_compare:nNtTF { \l_tmpb_tl } = { \c@jCol + 1 }
6243 {
6244 \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6245 { \bool_set_false:N \g_tmpa_bool }
6246 }
6247 {
6248 \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6249 {

```

```

6250         \int_compare:nNnTF { \l_tmpb_tl } = { \c_one_int }
6251         { \bool_set_false:N \g_tmpa_bool }
6252         {
6253             \@@_if_in_corner:nT
6254             { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6255             { \bool_set_false:N \g_tmpa_bool }
6256         }
6257     }
6258 }
6259 }

```

```

6260 \cs_new_protected:Npn \@@_vline_ii:
6261 {
6262     \tl_clear:N \l_@@_tikz_rule_tl
6263     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6264     \bool_if:NTF \l_@@_dotted_bool
6265     { \@@_vline_iv: }
6266     {
6267         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6268         { \@@_vline_iii: }
6269         { \@@_vline_v: }
6270     }
6271 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

6272 \cs_new_protected:Npn \@@_vline_iii:
6273 {
6274     \pgfpicture
6275     \pgfrememberpicturepositiononpagetrue
6276     \pgf@relevantforpicturesizefalse
6277     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6278     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6279     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6280     \dim_set:Nn \l_tmpb_dim
6281     {
6282         \pgf@x
6283         - 0.5 \l_@@_rule_width_dim
6284         +
6285         ( \arrayrulewidth * \l_@@_multiplicity_int
6286           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6287     }
6288     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6289     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6290     \bool_lazy_all:nT
6291     {
6292         { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6293         { \cs_if_exist_p:N \CT@drsc@ }
6294         { ! \tl_if_blank_p:o \CT@drsc@ }
6295     }
6296     {
6297         \group_begin:
6298         \CT@drsc@
6299         \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6300         \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6301         \dim_set:Nn \l_@@_tmpd_dim
6302         {
6303             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6304             * ( \l_@@_multiplicity_int - 1 )
6305         }
6306         \pgfpathrectanglecorners
6307         { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6308         { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }

```

```

6309     \pgfusepath { fill }
6310     \group_end:
6311 }
6312 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6313 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6314 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6315 {
6316     \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6317     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6318     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6319 }
6320 \CT@arc@
6321 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6322 \pgfsetrectcap
6323 \pgfusepathqstroke
6324 \endpgfpicture
6325 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6326 \cs_new_protected:Npn \@@_vline_iv:
6327 {
6328     \pgfpicture
6329     \pgfrememberpicturepositiononpagetrue
6330     \pgf@relevantforpicturesizefalse
6331     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6332     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6333     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6334     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6335     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6336     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6337     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6338     \CT@arc@
6339     \@@_draw_line:
6340     \endpgfpicture
6341 }

```

The following code is for the case when the user uses the key `tikz`.

```

6342 \cs_new_protected:Npn \@@_vline_v:
6343 {
6344     \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).

```

6345     \CT@arc@
6346     \tl_if_empty:NF \l_@@_rule_color_tl
6347     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6348     \pgfrememberpicturepositiononpagetrue
6349     \pgf@relevantforpicturesizefalse
6350     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6351     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6352     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6353     \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6354     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6355     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6356     \exp_args:No \tikzset \l_@@_tikz_rule_tl
6357     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6358     ( \l_tmpb_dim , \l_tmpa_dim ) --
6359     ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6360     \end { tikzpicture }
6361 }

```


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).

```

6362 \cs_new_protected:Npn \@@_draw_vlines:
6363 {
6364   \int_step_inline:nnn
6365   {
6366     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6367     { 2 }
6368     { 1 }
6369   }
6370   {
6371     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6372     { \c@jCol }
6373     { \int_eval:n { \c@jCol + 1 } }
6374   }
6375   {
6376     \str_if_eq:eeF { \l_@@_vlines_clist } { all }
6377     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6378     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6379   }
6380 }

```

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}`.

```

6381 \cs_new_protected:Npn \@@_hline:n #1
6382 {

```

The group is for the options.

```

6383   \group_begin:
6384   \int_set_eq:NN \l_@@_end_int \c@jCol
6385   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6386   \@@_hline_i:
6387   \group_end:
6388 }
6389 \cs_new_protected:Npn \@@_hline_i:
6390 {

```

`\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 `\Block` or a virtual block corresponding to a dotted line, created by `\Cdots`, `\Vdots`, etc.), we will set `\g_tmpa_bool` to `false` and the small horizontal rule won't be drawn.

```

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:nNtT { \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:nNtT { \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:nNtTF { \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:nNtTF { \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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6507 \cs_new_protected:Npn \@@_hline_iv:
6508 {
6509   \pgfpicture
6510   \pgfrememberpicturepositiononpagetrue
6511   \pgf@relevantforpicturesizefalse
6512   \@@_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   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6516   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6517   \int_compare:nNt { \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   \@@_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:nNt { \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:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
6610   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
6611   {
6612     \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }

```

We delete the last character which is a space.

```

6613     \str_set:Ne \l_@@_command_str
6614     { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
6615   }
6616   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
6617   {
6618     \str_set:Ne \l_@@_ccommand_str
6619     { \str_tail:N \l_@@_ccommand_str }
6620     \str_set:Ne \l_@@_ccommand_str
6621     { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
6622   }

```

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.

```

6623   \bool_lazy_all:nTF
6624   {
6625     { \str_if_empty_p:N \l_@@_letter_str }
6626     { \str_if_empty_p:N \l_@@_command_str }
6627     { \str_if_empty_p:N \l_@@_ccommand_str }
6628   }
6629   { \@@_error:n { No~letter~and~no~command } }
6630   { \@@_custom_line_i:o \l_@@_other_keys_tl }
6631 }

6632 \keys_define:nn { nicematrix / custom-line }
6633 {
6634   letter .str_set:N = \l_@@_letter_str ,
6635   letter .value_required:n = true ,
6636   command .str_set:N = \l_@@_command_str ,
6637   command .value_required:n = true ,
6638   ccommand .str_set:N = \l_@@_ccommand_str ,
6639   ccommand .value_required:n = true ,
6640 }

```

```

6641 \cs_new_protected:Npn \@@_custom_line_i:n #1
6642 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6643   \bool_set_false:N \l_@@_tikz_rule_bool
6644   \bool_set_false:N \l_@@_dotted_rule_bool
6645   \bool_set_false:N \l_@@_color_bool

```

```

6646 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6647 \bool_if:NT \l_@@_tikz_rule_bool
6648 {
6649   \IfPackageLoadedF { tikz }
6650   { \@@_error:n { tikz-in-custom-line-without-tikz } }
6651   \bool_if:NT \l_@@_color_bool
6652   { \@@_error:n { color-in-custom-line-with-tikz } }
6653 }
6654 \bool_if:NT \l_@@_dotted_rule_bool
6655 {
6656   \int_compare:nNnT { \l_@@_multiplicity_int } > { \c_one_int }
6657   { \@@_error:n { key-multiplicity-with-dotted } }
6658 }
6659 \str_if_empty:NF \l_@@_letter_str
6660 {
6661   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6662   { \@@_error:n { Several-letters } }
6663   {
6664     \tl_if_in:NoTF
6665     \c_@@_forbidden_letters_str
6666     \l_@@_letter_str
6667     { \@@_error:ne { Forbidden-letter } \l_@@_letter_str }
6668   }

```

During the analysis of the preamble provided by the final user, our automaton, for the letter corresponding at the custom line, will directly use the following command that you define in the main hash table of TeX.

```

6669           \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6670           { \@@_v_custom_line:nn { #1 } }
6671       }
6672   }
6673 }
6674 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6675 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6676 }
6677 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6678 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6679 \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.

```

6680 \keys_define:nn { nicematrix / custom-line-bis }
6681 {
6682   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6683   multiplicity .initial:n = 1 ,
6684   multiplicity .value_required:n = true ,
6685   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6686   color .value_required:n = true ,
6687   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6688   tikz .value_required:n = true ,
6689   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6690   dotted .value_forbidden:n = true ,
6691   total-width .code:n = { } ,
6692   total-width .value_required:n = true ,
6693   width .code:n = { } ,
6694   width .value_required:n = true ,
6695   sep-color .code:n = { } ,
6696   sep-color .value_required:n = true ,
6697   unknown .code:n =
6698     \@@_unknown_key:nn
6699     { nicematrix / custom-line-bis }

```

```

6700         { Unknown-key-for-custom-line }
6701     }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6702 \bool_new:N \l_@@_dotted_rule_bool
6703 \bool_new:N \l_@@_tikz_rule_bool
6704 \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`.

```

6705 \keys_define:nn { nicematrix / custom-line-width }
6706 {
6707     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6708     multiplicity .initial:n = 1 ,
6709     multiplicity .value_required:n = true ,
6710     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6711     total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6712                     \bool_set_true:N \l_@@_total_width_bool ,
6713     total-width .value_required:n = true ,
6714     width .meta:n = { total-width = #1 } ,
6715     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6716 }

```

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`).

```

6717 \cs_new_protected:Npn \@@_h_custom_line:n #1
6718 {

```

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`).

```

6719     \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6720     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6721 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule on only some of the columns of the array (hence the letter `c` as in `\cline`). `#1` is the whole set of keys to pass to the command `\@@_hline:n` (which is in the internal `\CodeAfter`).

```

6722 \cs_new_protected:Npn \@@_c_custom_line:n #1
6723 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

6724     \exp_args:Nc \NewExpandableDocumentCommand
6725     { nicematrix - \l_@@_ccommand_str }
6726     { 0 { } m }
6727     {
6728         \noalign
6729         {
6730             \@@_compute_rule_width:n { #1 , ##1 }
6731             \skip_vertical:n { \l_@@_rule_width_dim }
6732             \clist_map_inline:nn
6733             { ##2 }
6734             { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6735         }
6736     }
6737     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6738 }

```


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*.

```

6739 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6740 {
6741   \tl_if_in:nnTF { #2 } { - }
6742   { \@@_cut_on_hyphen:w #2 \q_stop }
6743   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6744   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6745   {
6746     \@@_hline:n
6747     {
6748       #1 ,
6749       start = \l_tmpa_tl ,
6750       end = \l_tmpb_tl ,
6751       position = \int_eval:n { \c@iRow + 1 } ,
6752       total-width = \dim_use:N \l_@@_rule_width_dim
6753     }
6754   }
6755 }

6756 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6757 {
6758   \bool_set_false:N \l_@@_tikz_rule_bool
6759   \bool_set_false:N \l_@@_total_width_bool
6760   \bool_set_false:N \l_@@_dotted_rule_bool
6761   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6762   \bool_if:NF \l_@@_total_width_bool
6763   {
6764     \bool_if:NTF \l_@@_dotted_rule_bool
6765     { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6766     {
6767       \bool_if:NF \l_@@_tikz_rule_bool
6768       {
6769         \dim_set:Nn \l_@@_rule_width_dim
6770         {
6771           \arrayrulewidth * \l_@@_multiplicity_int
6772           + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6773         }
6774       }
6775     }
6776   }
6777 }

```

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

```

6778 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
6779 {
6780   \str_if_eq:nnTF { #2 } { [ ] }
6781   { \@@_v_custom_line_i:nw { #1 } [ ] }
6782   { \@@_v_custom_line_ii:nn { #2 } { #1 } }
6783 }
6784 \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
6785 { \@@_v_custom_line:nn { #1 , #2 } }
6786 \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
6787 {
6788   \@@_compute_rule_width:n { #2 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6789 \tl_gput_right:Ne \g_@@_array_preamble_tl
6790 { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6791 \tl_gput_right:Ne \g_@@_pre_code_after_tl
6792 {
6793   \@@_vline:n
6794   {

```

```

6795         #2 ,
6796         position = \int_eval:n { \c@jCol + 1 } ,
6797         total-width = \dim_use:N \l_@@_rule_width_dim
6798     }
6799 }
6800 \@@_rec_preamble:n #1
6801 }
6802 \@@_custom_line:n
6803 { 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.

```

6804 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6805 {
6806     \int_compare:nNnT { \l_tmpa_tl } > { #1 }
6807     {
6808         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6809         {
6810             \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6811             {
6812                 \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6813                 { \bool_gset_false:N \g_tmpa_bool }
6814             }
6815         }
6816     }
6817 }

```

The same for vertical rules.

```

6818 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6819 {
6820     \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6821     {
6822         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6823         {
6824             \int_compare:nNnT { \l_tmpb_tl } > { #2 }
6825             {
6826                 \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6827                 { \bool_gset_false:N \g_tmpa_bool }
6828             }
6829         }
6830     }
6831 }
6832 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6833 {
6834     \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6835     {
6836         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6837         {
6838             \int_compare:nNnTF { \l_tmpa_tl } = { #1 }
6839             { \bool_gset_false:N \g_tmpa_bool }
6840             {
6841                 \int_compare:nNnT { \l_tmpa_tl } = { #3 + 1 }
6842                 { \bool_gset_false:N \g_tmpa_bool }
6843             }
6844         }
6845     }
6846 }

```

```

6847 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6848 {
6849   \int_compare:nNtT { \l_tmpa_tl } > { #1 - 1 }
6850   {
6851     \int_compare:nNtT { \l_tmpa_tl } < { #3 + 1 }
6852     {
6853       \int_compare:nNtTF { \l_tmpb_tl } = { #2 }
6854       { \bool_gset_false:N \g_tmpa_bool }
6855       {
6856         \int_compare:nNtT { \l_tmpb_tl } = { #4 + 1 }
6857         { \bool_gset_false:N \g_tmpa_bool }
6858       }
6859     }
6860   }
6861 }

```

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.

```

6862 \cs_new_protected:Npn \@@_compute_corners:
6863 {
6864   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6865   { \@@_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`).

```

6866   \clist_clear:N \l_@@_corners_cells_clist
6867   \clist_map_inline:Nn \l_@@_corners_cells_clist
6868   {
6869     \str_case:nnF { ##1 }
6870     {
6871       { NW }
6872       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6873       { NE }
6874       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6875       { SW }
6876       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6877       { SE }
6878       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6879     }
6880     { \@@_error:nn { bad~corner } { ##1 } }
6881   }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6882   \clist_if_empty:NF \l_@@_corners_cells_clist
6883   {

```

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.

```

6884     \tl_gput_right:Ne \g_@@_aux_tl
6885     {
6886       \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
6887       { \l_@@_corners_cells_clist }
6888     }
6889   }
6890 }

```

```

6891 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6892 {
6893   \int_step_inline:nnn { #1 } { #3 }
6894   {
6895     \int_step_inline:nnn { #2 } { #4 }
6896     { \cs_set_nopar:cpn { @@ _ block _ ##1 - ###1 } { } }
6897   }
6898 }

6899 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6900 {
6901   \cs_if_exist:cTF
6902   { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6903   { \prg_return_true: }
6904   { \prg_return_false: }
6905 }

```

“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.

```

6906 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6907 {

```

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.

```

6908   \bool_set_false:N \l_tmpa_bool
6909   \int_zero_new:N \l_@@_last_empty_row_int
6910   \int_set:Nn \l_@@_last_empty_row_int { #1 }
6911   \int_step_inline:nnnn { #1 } { #3 } { #5 }
6912   {
6913     \bool_lazy_or:nnTF
6914     {
6915       \cs_if_exist_p:c
6916       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6917     }
6918     { \@@_if_in_block_p:nn { ##1 } { #2 } }
6919     { \bool_set_true:N \l_tmpa_bool }
6920     {
6921       \bool_if:NF \l_tmpa_bool
6922       { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6923     }
6924   }

```

Now, you determine the last empty cell in the row of number 1.

```

6925   \bool_set_false:N \l_tmpa_bool
6926   \int_zero_new:N \l_@@_last_empty_column_int
6927   \int_set:Nn \l_@@_last_empty_column_int { #2 }
6928   \int_step_inline:nnnn { #2 } { #4 } { #6 }
6929   {
6930     \bool_lazy_or:nnTF
6931     {

```

```

6932     \cs_if_exist_p:c
6933     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6934   }
6935   { \@@_if_in_block_p:nn { #1 } { ##1 } }
6936   { \bool_set_true:N \l_tmpa_bool }
6937   {
6938     \bool_if:NF \l_tmpa_bool
6939     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6940   }
6941 }

```

Now, we loop over the rows.

```

6942   \int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int }
6943   {

```

We treat the row number `##1` with another loop.

```

6944     \bool_set_false:N \l_tmpa_bool
6945     \int_step_inline:nnnn { #2 } { #4 } { \l_@@_last_empty_column_int }
6946     {
6947       \bool_lazy_or:nnTF
6948       { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6949       { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6950       { \bool_set_true:N \l_tmpa_bool }
6951     {
6952       \bool_if:NF \l_tmpa_bool
6953       {
6954         \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6955         \clist_put_right:Nn
6956         \l_@@_corners_cells_clist
6957         { ##1 - #####1 }
6958         \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6959       }
6960     }
6961   }
6962 }
6963 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6964 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6965 \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.

```

6966 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

6967 \keys_define:nn { nicematrix / NiceMatrixBlock }
6968 {
6969   auto-columns-width .code:n =
6970   {
6971     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6972     \dim_gzero_new:N \g_@@_max_cell_width_dim
6973     \bool_set_true:N \l_@@_auto_columns_width_bool
6974   }
6975 }

```

```

6976 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6977 {
6978   \int_gincr:N \g_@@_NiceMatrixBlock_int
6979   \dim_zero:N \l_@@_columns_width_dim
6980   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6981   \bool_if:NT \l_@@_block_auto_columns_width_bool
6982   {
6983     \cs_if_exist:cT
6984     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6985     {
6986       \dim_set:Nn \l_@@_columns_width_dim
6987       {
6988         \use:c
6989         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6990       }
6991     }
6992   }
6993 }

```

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`).

```

6994 {
6995   \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.

```

6996   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6997   {
6998     \bool_if:NT \l_@@_block_auto_columns_width_bool
6999     {
7000       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
7001       \iow_shipout:Ne \@mainaux
7002       {
7003         \cs_gset:cpn
7004         { @@ _ 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.

```

7005         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7006       }
7007       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
7008     }
7009   }
7010   \ignorespacesafterend
7011 }

```

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).

```

7012 \cs_new_protected:Npn \@@_create_extra_nodes:
7013 {
7014   \bool_if:nTF \l_@@_medium_nodes_bool
7015   {
7016     \bool_if:NTF \l_@@_no_cell_nodes_bool
7017     { \@@_error:n { extra-nodes~with~no~cell~nodes } }
7018     {
7019       \bool_if:NTF \l_@@_large_nodes_bool

```

```

7020         \@@_create_medium_and_large_nodes:
7021         \@@_create_medium_nodes:
7022     }
7023 }
7024 {
7025     \bool_if:NT \l_@@_large_nodes_bool
7026     {
7027         \bool_if:NTF \l_@@_no_cell_nodes_bool
7028         { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7029         \@@_create_large_nodes:
7030     }
7031 }
7032 }

```

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`.

```

7033 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7034 {
7035     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7036     {
7037         \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7038         \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7039         \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7040         \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7041     }
7042     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7043     {
7044         \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7045         \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7046         \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7047         \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7048     }

```

We begin the two nested loops over the rows and the columns of the array.

```

7049     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7050     {
7051         \int_step_variable:nnNn
7052         \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.

```

7053     {
7054         \cs_if_exist:cT
7055         { 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`.

```

7056     {
7057         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7058         \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7059         { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7060         \seq_if_in:Nef \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7061         {
7062             \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7063             { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7064         }

```

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`.

```

7065         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7066         \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7067         { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } { \pgf@y } }
7068         \seq_if_in:Nef \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7069         {
7070             \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7071             { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } { \pgf@x } }
7072         }
7073     }
7074 }
7075 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7076 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7077 {
7078     \dim_compare:nNnT
7079     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7080     {
7081         \@@_qpoint:n { row - \@@_i: - base }
7082         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7083         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7084     }
7085 }
7086 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7087 {
7088     \dim_compare:nNnT
7089     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7090     {
7091         \@@_qpoint:n { col - \@@_j: }
7092         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7093         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7094     }
7095 }
7096 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7097 \cs_new_protected:Npn \@@_create_medium_nodes:
7098 {
7099     \pgfpicture
7100     \pgfrememberpicturepositiononpagetrue
7101     \pgf@relevantforpicturesizefalse
7102     \@@_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”.

```

7103     \tl_set:Nn \l_@@_suffix_tl { -medium }
7104     \@@_create_nodes:

```



```

7105 \endpgfpicture
7106 }

```

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:`.

```

7107 \cs_new_protected:Npn \@@_create_large_nodes:
7108 {
7109   \pgfpicture
7110   \pgfrememberpicturepositiononpagetrue
7111   \pgf@relevantforpicturesizefalse
7112   \@@_computations_for_medium_nodes:
7113   \@@_computations_for_large_nodes:
7114   \tl_set:Nn \l_@@_suffix_tl { - large }
7115   \@@_create_nodes:
7116 \endpgfpicture
7117 }
7118 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7119 {
7120   \pgfpicture
7121   \pgfrememberpicturepositiononpagetrue
7122   \pgf@relevantforpicturesizefalse
7123   \@@_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”.

```

7124   \tl_set:Nn \l_@@_suffix_tl { - medium }
7125   \@@_create_nodes:
7126   \@@_computations_for_large_nodes:
7127   \tl_set:Nn \l_@@_suffix_tl { - large }
7128   \@@_create_nodes:
7129 \endpgfpicture
7130 }

```

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.

```

7131 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7132 {
7133   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7134   \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`.

```

7135   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7136   {
7137     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7138     {
7139       (
7140         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7141         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7142       )
7143       / 2
7144     }
7145     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7146     { l_@@_row _ \@@_i: _ min _ dim }
7147   }
7148   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:

```

¹⁵If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7149 {
7150   \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7151   {
7152     (
7153       \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7154       \dim_use:c
7155         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7156     )
7157     / 2
7158   }
7159   \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7160   { l_@@_column _ \@@_j: _ max _ dim }
7161 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7162   \dim_sub:cn
7163     { l_@@_column _ 1 _ min _ dim }
7164   \l_@@_left_margin_dim
7165   \dim_add:cn
7166     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7167   \l_@@_right_margin_dim
7168 }

```

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).

```

7169 \cs_new_protected:Npn \@@_create_nodes:
7170 {
7171   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7172   {
7173     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7174     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7175       \@@_pgf_rect_node:nnnnn
7176       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7177       { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7178       { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7179       { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7180       { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7181       \str_if_empty:NF \l_@@_name_str
7182       {
7183         \pgfnodealias
7184         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7185         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7186       }
7187     }
7188   }
7189   \int_step_inline:nn { \c@iRow }
7190   {
7191     \pgfnodealias
7192     { \@@_env: - ##1 - last \l_@@_suffix_tl }
7193     { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7194   }
7195   \int_step_inline:nn { \c@jCol }
7196   {
7197     \pgfnodealias
7198     { \@@_env: - last - ##1 \l_@@_suffix_tl }
7199     { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7200   }

```

```

7201 \pgfnodealias % added 2025-04-05
7202 { \@@_env: - last - last \l_@@_suffix_tl }
7203 { \@@_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 .

```

7204 \seq_map_pairwise_function:NNN
7205 \g_@@_multicolumn_cells_seq
7206 \g_@@_multicolumn_sizes_seq
7207 \@@_node_for_multicolumn:nn
7208 }

```

```

7209 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7210 {
7211   \cs_set_nopar:Npn \@@_i: { #1 }
7212   \cs_set_nopar:Npn \@@_j: { #2 }
7213 }

```

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”).

```

7214 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7215 {
7216   \@@_extract_coords_values: #1 \q_stop
7217   \@@_pgf_rect_node:nnnnn
7218   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7219   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7220   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7221   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: + #2 - 1 } _ max _ dim } }
7222   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7223   \str_if_empty:NF \l_@@_name_str
7224   {
7225     \pgfnodealias
7226     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7227     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7228   }
7229 }

```

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).

```

7230 \keys_define:nn { nicematrix / Block / FirstPass }
7231 {
7232   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7233           \bool_set_true:N \l_@@_p_block_bool ,
7234   j .value_forbidden:n = true ,
7235   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7236   l .value_forbidden:n = true ,
7237   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7238   r .value_forbidden:n = true ,
7239   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7240   c .value_forbidden:n = true ,

```

```

7241 L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7242 L .value_forbidden:n = true ,
7243 R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7244 R .value_forbidden:n = true ,
7245 C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7246 C .value_forbidden:n = true ,
7247 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7248 t .value_forbidden:n = true ,
7249 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7250 T .value_forbidden:n = true ,
7251 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7252 b .value_forbidden:n = true ,
7253 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7254 B .value_forbidden:n = true ,
7255 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7256 m .value_forbidden:n = true ,
7257 v-center .meta:n = m ,
7258 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7259 p .value_forbidden:n = true ,
7260 color .code:n =
7261   \@@_color:n { #1 }
7262   \tl_set_rescan:Nnn
7263     \l_@@_draw_tl
7264     { \char_set_catcode_other:N ! }
7265     { #1 } ,
7266 color .value_required:n = true ,
7267 respect-arraystretch .code:n =
7268   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7269 respect-arraystretch .value_forbidden:n = true ,
7270 }

```

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.

```

7271 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7272 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7273 {

```

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).

```

7274   \tl_if_blank:nTF { #2 }
7275   { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7276   {
7277     \tl_if_in:nnTF { #2 } { - }
7278     {
7279       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7280       \@@_Block_i_czech:w \@@_Block_i:w
7281       #2 \q_stop
7282     }
7283     {
7284       \@@_error:nn { Bad~argument~for~Block } { #2 }
7285       \@@_Block_ii:nnnnn \c_one_int \c_one_int
7286     }
7287   }
7288   { #1 } { #3 } { #4 }
7289   \ignorespaces
7290 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7291 \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`.

```

7292 {
7293   \char_set_catcode_active:N -
7294   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7295 }

```

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.

```

7296 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7297 {

```

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).

```

7298   \bool_lazy_or:nnTF
7299     { \tl_if_blank_p:n { #1 } }
7300     { \str_if_eq_p:ee { * } { #1 } }
7301     { \int_set:Nn \l_tmpa_int { 100 } }
7302     { \int_set:Nn \l_tmpa_int { #1 } }
7303   \bool_lazy_or:nnTF
7304     { \tl_if_blank_p:n { #2 } }
7305     { \str_if_eq_p:ee { * } { #2 } }
7306     { \int_set:Nn \l_tmpb_int { 100 } }
7307     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7308   \int_compare:nNnTF { \l_tmpb_int } = { \c_one_int }
7309   {
7310     \tl_if_empty:NNTF \l_@@_hpos_cell_tl
7311       { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7312       { \str_set:Nn \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7313   }
7314   { \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.

```

7315   \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7316   \tl_set:Nn \l_tmpa_tl
7317   {
7318     { \int_use:N \c@iRow }
7319     { \int_use:N \c@jCol }
7320     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7321     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7322   }

```

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).

```

7323   \bool_set_false:N \l_tmpa_bool
7324   \bool_if:NT \l_@@_amp_in_blocks_bool

```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

7325     { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7326     \bool_case:nF
7327     {
7328         \l_tmpa_bool                                { \@@_Block_vii:eennn }
7329         \l_@@_p_block_bool                          { \@@_Block_v:eennn }

```

For the blocks mono-column, we will compose right away in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a X column, we should not do that since the width is determined by another way. This should be the same for the p, m and b columns and we should modify that point. However, for the X column, it's imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

7330         \l_@@_X_bool                                { \@@_Block_v:eennn }
7331         { \tl_if_empty_p:n { #5 } }                  { \@@_Block_v:eennn }
7332         { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7333         { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7334     }
7335     { \@@_Block_v:eennn }
7336     { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7337 }

```

The following macro is for the case of a `\Block` which is mono-row or mono-column (or both) and don't use the key p. In that case, the content of the block is composed right away in a box (because we have to take into account the dimensions of that box for the width of the current column or the height and the depth of the current row). However, that box will be put in the array *after the construction of the array* (by using PGF) with `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn` which will do the main job.

#1 is *i* (the number of rows of the block), #2 is *j* (the number of columns of the block), #3 is the list of key=values pairs, #4 are the tokens to put before the potential math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7338 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7339 {
7340     \int_gincr:N \g_@@_block_box_int
7341     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7342     {
7343         \tl_gput_right:Ne \g_@@_pre_code_after_tl
7344         {
7345             \@@_actually_diagbox:nnnnnn
7346             { \int_use:N \c@iRow }
7347             { \int_use:N \c@jCol }
7348             { \int_eval:n { \c@iRow + #1 - 1 } }
7349             { \int_eval:n { \c@jCol + #2 - 1 } }
7350             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7351             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7352         }
7353     }
7354     \box_gclear_new:c
7355     { g_@@_block_box _ \int_use:N \g_@@_block_box_int _ box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7356     \hbox_gset:cn
7357     { g_@@_block_box _ \int_use:N \g_@@_block_box_int _ box }
7358     {

```

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`).

```

7359     \tl_if_empty:NTF \l_@@_color_tl

```

```

7360      { \int_compare:nNt { #2 } = { \c_one_int } { \set@color } }
7361      { \l_@@_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`.

```

7362      \int_compare:nNt { #1 } = { \c_one_int }
7363      {
7364          \int_if_zero:nTF { \c_iRow }
7365          {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7366      \cs_set_eq:NN \Block \l_@@_NullBlock:
7367      \l_@@_code_for_first_row_tl
7368      }
7369      {
7370          \int_compare:nNt { \c_iRow } = { \l_@@_last_row_int }
7371          {
7372              \cs_set_eq:NN \Block \l_@@_NullBlock:
7373              \l_@@_code_for_last_row_tl
7374          }
7375      }
7376      \g_@@_row_style_tl
7377  }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7378      \l_@@_reset_arraystretch:
7379      \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7380      #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.).

```

7381      \l_@@_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}`.

```

7382      \bool_if:NTF \l_@@_tabular_bool
7383      {

```

```

7384         \bool_lazy_all:nTF
7385         {
7386             { \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.

```

7387         {
7388             ! \dim_compare_p:nNn
7389             { \l_@@_col_width_dim } < { \c_zero_dim }
7390         }
7391         { ! \g_@@_rotate_bool }
7392     }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7393     {
7394         \use:e
7395     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7396         \exp_not:N \begin { minipage }
7397         [ \str_lowercase:f \l_@@_vpos_block_str ]
7398         { \l_@@_col_width_dim }
7399         \str_case:on \l_@@_hpos_block_str
7400         { c \centering r \raggedleft l \raggedright }
7401     }
7402     #5
7403     \end { minipage }
7404 }

```

In the other cases, we use a `{tabular}`.

```

7405     {
7406         \use:e
7407     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7408         \exp_not:N \begin { tabular }
7409         [ \str_lowercase:f \l_@@_vpos_block_str ]
7410         { @ { } \l_@@_hpos_block_str @ { } }
7411     }
7412     #5
7413     \end { tabular }
7414 }
7415 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7416     {
7417         $ % $
7418         \use:e
7419     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7420         \exp_not:N \begin { array }
7421         [ \str_lowercase:f \l_@@_vpos_block_str ]
7422         { @ { } \l_@@_hpos_block_str @ { } }
7423     }
7424     #5
7425     \end { array }
7426     $ % $
7427 }
7428 }

```

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).

```

7429     \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.

```

7430     \int_compare:nNnT { #2 } = { \c_one_int }
7431     {
7432         \dim_gset:Nn \g_@@_blocks_wd_dim
7433         {
7434             \dim_max:nn
7435             { \g_@@_blocks_wd_dim }
7436             {
7437                 \box_wd:c
7438                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7439             }
7440         }
7441     }

```

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.

```

7442     \int_compare:nNnT { #1 } = { \c_one_int }
7443     {
7444         \bool_lazy_any:nT
7445         {
7446             { \str_if_empty_p:N \l_@@_vpos_block_str }
7447             { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
7448             { \str_if_eq_p:ee { \l_@@_vpos_block_str } { b } }
7449         }
7450         { \@@_adjust_blocks_ht_dp: }
7451     }
7452     \seq_gput_right:Ne \g_@@_blocks_seq
7453     {
7454         \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.

```

7455     {
7456         \exp_not:n { #3 } ,
7457         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7458         \bool_if:NT \g_@@_rotate_bool
7459         {
7460             \bool_if:NTF \g_@@_rotate_c_bool
7461             { m }
7462             {
7463                 \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7464                 { T }
7465             }
7466         }
7467     }
7468     {
7469         \box_use_drop:c
7470         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7471     }
7472 }
7473 \bool_set_false:N \g_@@_rotate_c_bool
7474 }

7475 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7476 {
7477     \dim_gset:Nn \g_@@_blocks_ht_dim
7478     {

```

```

7479     \dim_max:nn
7480     { \g_@@_blocks_ht_dim }
7481     {
7482         \box_ht:c
7483         { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7484     }
7485 }
7486 \dim_gset:Nn \g_@@_blocks_dp_dim
7487 {
7488     \dim_max:nn
7489     { \g_@@_blocks_dp_dim }
7490     {
7491         \box_dp:c
7492         { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7493     }
7494 }
7495 }

7496 \cs_new:Npn \@@_adjust_hpos_rotate:
7497 {
7498     \bool_if:NT \g_@@_rotate_bool
7499     {
7500         \str_set:Ne \l_@@_hpos_block_str
7501         {
7502             \bool_if:NTF \g_@@_rotate_c_bool
7503             { c }
7504             {
7505                 \str_case:onF \l_@@_vpos_block_str
7506                 { b l B l t r T r }
7507                 {
7508                     \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
7509                     { r }
7510                     { l }
7511                 }
7512             }
7513         }
7514     }
7515 }
7516 \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*.

```

7517 \cs_new_protected:Npn \@@_rotate_box_of_block:
7518 {
7519     \box_grotate:cn
7520     { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7521     { 90 }
7522     \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7523     {
7524         \vbox_gset_top:cn
7525         { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7526         {
7527             \skip_vertical:n { 0.8 ex }
7528             \box_use:c
7529             { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7530         }
7531     }
7532     \bool_if:NT \g_@@_rotate_c_bool
7533     {
7534         \hbox_gset:cn
7535         { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7536         {

```

```

7537         $ % $
7538         \vcenter
7539         {
7540             \box_use:c
7541             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7542         }
7543         $ % $
7544     }
7545 }
7546 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key `p`). In that case, the content of the block is *not* composed right away in a box. The composition in a box will be done further, just after the construction of the array (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnn`).

#1 is i (the number of rows of the block), **#2** is j (the number of columns of the block), **#3** is the list of `key=values` pairs, **#4** are the tokens to put before the math mode and before the composition of the block and **#5** is the label (=content) of the block.

```

7547 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7548 {
7549     \seq_gput_right:Ne \g_@@_blocks_seq
7550     {
7551         \l_tmpa_tl
7552         { \exp_not:n { #3 } }
7553         {
7554             \bool_if:NTF \l_@@_tabular_bool
7555             {
7556                 \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7557     \@@_reset_arraystretch:
7558     \exp_not:n
7559     {
7560         \dim_zero:N \extrarowheight
7561         #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).

```

7562         \tag_if_active:T { \tag_stop:n { table } }
7563         \use:e
7564         {
7565             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7566             { @ { } \l_@@_hpos_block_str @ { } }
7567         }
7568         #5
7569         \end { tabular }
7570     }
7571     \group_end:
7572 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7573     {
7574         \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7575     \@@_reset_arraystretch:
7576     \exp_not:n
7577     {
7578         \dim_zero:N \extrarowheight
7579         #4
7580         $ % $

```

```

7581         \use:e
7582         {
7583             \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7584             { @ { } \l_@@_hpos_block_str @ { } }
7585         }
7586         #5
7587         \end { array }
7588         $ % $
7589     }
7590     \group_end:
7591 }
7592 }
7593 }
7594 }
7595 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7596 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7597 {
7598     \seq_gput_right:Ne \g_@@_blocks_seq
7599     {
7600         \l_tmpa_tl
7601         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7602         { { \exp_not:n { #4 #5 } } }
7603     }
7604 }
7605 \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`.

```

7606 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7607 {
7608     \seq_gput_right:Ne \g_@@_blocks_seq
7609     {
7610         \l_tmpa_tl
7611         { \exp_not:n { #3 } }
7612         { \exp_not:n { #4 #5 } }
7613     }
7614 }
7615 \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).

```

7616 \keys_define:nn { nicematrix / Block / SecondPass }
7617 {
7618     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7619     ampersand-in-blocks .default:n = true ,
7620     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7621     tikz .code:n =
7622         \IfPackageLoadedTF { tikz }
7623         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7624         { \@@_error:n { tikz-key-without-tikz } } ,
7625     tikz .value_required:n = true ,
7626     fill .code:n =
7627         \tl_set_rescan:Nnn
7628         \l_@@_fill_tl
7629         { \char_set_catcode_other:N ! }

```

```

7630     { #1 } ,
7631 fill .value_required:n = true ,
7632 opacity .tl_set:N = \l_@@_opacity_tl ,
7633 opacity .value_required:n = true ,
7634 draw .code:n =
7635     \tl_set_rescan:Nnn
7636     \l_@@_draw_tl
7637     { \char_set_catcode_other:N ! }
7638     { #1 } ,
7639 draw .default:n = default ,
7640 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7641 rounded-corners .default:n = 4 pt ,
7642 color .code:n =
7643     \@@_color:n { #1 }
7644     \tl_set_rescan:Nnn
7645     \l_@@_draw_tl
7646     { \char_set_catcode_other:N ! }
7647     { #1 } ,
7648 borders .clist_set:N = \l_@@_borders_clist ,
7649 borders .value_required:n = true ,
7650 hvlines .meta:n = { vlines , hlines } ,
7651 vlines .bool_set:N = \l_@@_vlines_block_bool ,
7652 vlines .default:n = true ,
7653 hlines .bool_set:N = \l_@@_hlines_block_bool ,
7654 hlines .default:n = true ,
7655 line-width .dim_set:N = \l_@@_line_width_dim ,
7656 line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7657 j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7658     \bool_set_true:N \l_@@_p_block_bool ,
7659 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7660 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7661 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7662 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7663     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7664 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7665     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7666 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7667     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7668 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7669 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7670 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7671 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7672 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7673 m .value_forbidden:n = true ,
7674 v-center .meta:n = m ,
7675 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7676 p .value_forbidden:n = true ,
7677 name .tl_set:N = \l_@@_block_name_str , % .str_set:N ?
7678 name .value_required:n = true ,
7679 name .initial:n = ,
7680 respect-arraystretch .code:n =
7681     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7682 respect-arraystretch .value_forbidden:n = true ,
7683 transparent .bool_set:N = \l_@@_transparent_bool ,
7684 transparent .default:n = true ,
7685 transparent .initial:n = false ,
7686 unknown .code:n =
7687     \@@_unknown_key:nn
7688     { nicematrix / Block / SecondPass }
7689     { Unknown-key-for-Block }
7690 }

```

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.

```

7691 \cs_new_protected:Npn \@@_draw_blocks:
7692 {
7693   \bool_if:NTF \c_@@_revtex_bool
7694     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7695     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7696   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7697 }
7698 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7699 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7700   \int_zero:N \l_@@_last_row_int
7701   \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”).

```

7702   \int_compare:nNnTF { #3 } > { 98 }
7703     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7704     { \int_set:Nn \l_@@_last_row_int { #3 } }
7705   \int_compare:nNnTF { #4 } > { 98 }
7706     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7707     { \int_set:Nn \l_@@_last_col_int { #4 } }
7708   \int_compare:nNnTF { \l_@@_last_col_int } > { \g_@@_col_total_int }
7709     {
7710       \bool_lazy_and:nnTF
7711         { \l_@@_preamble_bool }
7712         {
7713           \int_compare_p:n
7714             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7715         }
7716       {
7717         \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7718         \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7719         \@@_msg_redirect_name:nn { columns-not-used } { none }
7720       }
7721       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7722     }
7723   {
7724     \int_compare:nNnTF { \l_@@_last_row_int } > { \g_@@_row_total_int }
7725       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7726       {
7727         \@@_Block_v:nneenn
7728           { #1 }
7729           { #2 }
7730           { \int_use:N \l_@@_last_row_int }
7731           { \int_use:N \l_@@_last_col_int }
7732           { #5 }
7733           { #6 }
7734       }
7735   }
7736 }

```

The following command `\@@_Block_v:nnnnnn` will actually draw the block. `#1` is the first row of the block; `#2` is the first column of the block; `#3` is the last row of the block; `#4` is the last column of the block; `#5` is a list of *key=value* options; `#6` is the label (content) of the block.

```

7737 \cs_new_protected:Npn \l_@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7738 {

```

The group is for the keys.

```

7739 \group_begin:
7740 \int_compare:nNt { #1 } = { #3 }
7741 { \str_set:Nn \l_@@_vpos_block_str { t } }
7742 \keys_set:nn { nicematrix / Block / SecondPass } { #5 }

```

If the content of the block contains &, we will have a special treatment (since the cell must be divided in several sub-cells). Remark that \tl_if_in:nnT is faster then \str_if_in:nnT.

```

7743 \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7744 \bool_lazy_and:nnT
7745 { \l_@@_vlines_block_bool }
7746 { ! \l_@@_ampersand_bool }
7747 {
7748 \tl_gput_right:Ne \g_nicematrix_code_after_tl
7749 {
7750 \l_@@_vlines_block:nnn
7751 { \exp_not:n { #5 } }
7752 { #1 - #2 }
7753 { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7754 }
7755 }
7756 \bool_if:NT \l_@@_hlines_block_bool
7757 {
7758 \tl_gput_right:Ne \g_nicematrix_code_after_tl
7759 {
7760 \l_@@_hlines_block:nnn
7761 { \exp_not:n { #5 } }
7762 { #1 - #2 }
7763 { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7764 }
7765 }
7766 \bool_if:NF \l_@@_transparent_bool
7767 {
7768 \bool_lazy_and:nnF { \l_@@_vlines_block_bool } { \l_@@_hlines_block_bool }
7769 {

```

The sequence of the positions of the blocks (excepted the blocks with the key hvlines) will be used when drawing the rules (in fact, there is also the \multicolumn and the \diagbox in that sequence).

```

7770 \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7771 { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7772 }
7773 }

```

```

7774 \tl_if_empty:NF \l_@@_draw_tl
7775 {
7776 \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7777 { \@@_error:n { hlines~with~color } }
7778 \tl_gput_right:Ne \g_nicematrix_code_after_tl
7779 {
7780 \l_@@_stroke_block:nnn

```

#5 are the options

```

7781 { \exp_not:n { #5 } }
7782 { #1 - #2 }
7783 { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7784 }
7785 \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7786 { { #1 } { #2 } { #3 } { #4 } }
7787 }

```

```

7788 \clist_if_empty:NF \l_@@_borders_clist
7789 {
7790   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7791   {
7792     \@@_stroke_borders_block:nnn
7793     { \exp_not:n { #5 } }
7794     { #1 - #2 }
7795     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7796   }
7797 }
7798 \tl_if_empty:NF \l_@@_fill_tl
7799 {
7800   \@@_add_opacity_to_fill:
7801   \tl_gput_right:Ne \g_@@_pre_code_before_tl
7802   {
7803     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7804     { #1 - #2 }
7805     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7806     { \dim_use:N \l_@@_rounded_corners_dim }
7807   }
7808 }
7809 \seq_if_empty:NF \l_@@_tikz_seq
7810 {
7811   \tl_gput_right:Ne \g_nicematrix_code_before_tl
7812   {
7813     \@@_block_tikz:nnnnn
7814     { \seq_use:Nn \l_@@_tikz_seq { , } }
7815     { #1 }
7816     { #2 }
7817     { \int_use:N \l_@@_last_row_int }
7818     { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7819   }
7820 }
7821 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7822 {
7823   \tl_gput_right:Ne \g_@@_pre_code_after_tl
7824   {
7825     \@@_actually_diagbox:nnnnnn
7826     { #1 }
7827     { #2 }
7828     { \int_use:N \l_@@_last_row_int }
7829     { \int_use:N \l_@@_last_col_int }
7830     { \exp_not:n { ##1 } }
7831     { \exp_not:n { ##2 } }
7832   }
7833 }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} & & one & \\\
& & two & \\\
three & four & five & \\\
six & seven & eight & \\\
\end{NiceTabular}

```


We highlight the node 1-1-block

our block		one
		two
three	four	five
six	seven	eight

We highlight the node 1-1-block-short

our block		one
		two
three	four	five
six	seven	eight

The construction of the node corresponding to the merged cells.

```

7834 \pgfpicture
7835 \pgfrememberpicturepositiononpagetrue
7836 \pgf@relevantforpicturesizefalse
7837 \@@_qpoint:n { row - #1 }
7838 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7839 \@@_qpoint:n { col - #2 }
7840 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7841 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7842 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7843 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7844 \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.

```

7845 \@@_pgf_rect_node:nnnnn
7846 { \@@_env: - #1 - #2 - block }
7847 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7848 \str_if_empty:NF \l_@@_block_name_str
7849 {
7850 \pgfnodealias
7851 { \@@_env: - \l_@@_block_name_str }
7852 { \@@_env: - #1 - #2 - block }
7853 \str_if_empty:NF \l_@@_name_str
7854 {
7855 \pgfnodealias
7856 { \l_@@_name_str - \l_@@_block_name_str }
7857 { \@@_env: - #1 - #2 - block }
7858 }
7859 }

```

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.

```

7860 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7861 {
7862 \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.

```

7863 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7864 {

```

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.

```

7865 \cs_if_exist:cT
7866 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7867 {
7868 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7869 {
7870 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7871 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7872 }
7873 }
7874 }

```

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.

```

7875 \dim_compare:nNnT { \l_tmpb_dim } = { \c_max_dim }
7876 {
7877   \@@_qpoint:n { col - #2 }
7878   \dim_set_eq:NN \l_tmpb_dim \pgf@x
7879 }
7880 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7881 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7882 {
7883   \cs_if_exist:cT
7884   { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7885   {
7886     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7887     {
7888       \pgfpointanchor
7889       { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7890       { east }
7891       \dim_set:Nn \l_@@_tmpd_dim
7892       { \dim_max:nn { \l_@@_tmpd_dim } { \pgf@x } }
7893     }
7894   }
7895 }
7896 \dim_compare:nNnT { \l_@@_tmpd_dim } = { - \c_max_dim }
7897 {
7898   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7899   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7900 }
7901 \@@_pgf_rect_node:nnnnn
7902 { \@@_env: - #1 - #2 - block - short }
7903 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7904 }

```

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.

```

7905 \bool_if:NT \l_@@_medium_nodes_bool
7906 {
7907   \@@_pgf_rect_node:nnn
7908   { \@@_env: - #1 - #2 - block - medium }
7909   { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7910   {
7911     \pgfpointanchor
7912     { \@@_env:
7913       - \int_use:N \l_@@_last_row_int
7914       - \int_use:N \l_@@_last_col_int - medium
7915     }
7916     { south-east }
7917   }
7918 }
7919 \endpgfpicture
7920

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand &.

```

7921 \bool_if:NNT \l_@@_ampersand_bool
7922 {
7923   \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7924   \int_zero_new:N \l_@@_split_int
7925   \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }

```

The following counters will be used to send information to `\cellcolor` if the user uses that command in a subcell.

```

7926 \int_zero_new:N \l_@@_first_row_int
7927 \int_zero_new:N \l_@@_first_col_int

```

```

7928 \int_zero_new:N \l_@@_last_row_int
7929 \int_zero_new:N \l_@@_last_col_int
7930 \int_set:Nn \l_@@_first_row_int { #1 }
7931 \int_set:Nn \l_@@_first_col_int { #2 }
7932 \int_set:Nn \l_@@_last_row_int { #3 }
7933 \int_set:Nn \l_@@_last_col_int { #4 }

7934 \pgfpicture
7935 \pgfrememberpicturepositiononpagetrue
7936 \pgf@relevantforpicturesizefalse
7937 \@@_qpoint:n { row - #1 }
7938 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7939 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7940 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7941 \@@_qpoint:n { col - #2 }
7942 \dim_set_eq:NN \l_tmpa_dim \pgf@x
7943 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7944 \dim_set:Nn \l_tmpb_dim
7945 { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7946 \bool_lazy_or:nnTF
7947 { \l_@@_vlines_block_bool }
7948 { \str_if_eq_p:ee { \l_@@_vlines_clist } { all } }
7949 {
7950 \int_step_inline:nn { \l_@@_split_int - 1 }
7951 {
7952 \pgfpathmoveto
7953 {
7954 \pgfpoint
7955 { \l_tmpa_dim + ##1 \l_tmpb_dim }
7956 \l_@@_tmpc_dim
7957 }
7958 \pgfpathlineto
7959 {
7960 \pgfpoint
7961 { \l_tmpa_dim + ##1 \l_tmpb_dim }
7962 \l_@@_tmpd_dim
7963 }
7964 \CT@arc@
7965 \pgfsetlinewidth { 1.1 \arrayrulewidth }
7966 \pgfsetrectcap
7967 \pgfusepathqstroke
7968 }
7969 }
7970 \cs_set_eq:NN \cellcolor \@@_subcellcolor
7971 \int_zero_new:N \l_@@_split_i_int

7972 \str_if_eq:eeTF { \l_@@_vpos_block_str } { T }
7973 {
7974 \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
7975 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7976 }
7977 {
7978 \str_if_eq:eeTF { \l_@@_vpos_block_str } { B }
7979 {
7980 \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
7981 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7982 }
7983 {
7984 \bool_lazy_or:nnTF
7985 { \int_compare_p:nNn { #1 } = { #3 } }
7986 { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
7987 {
7988 \@@_qpoint:n { row - #1 - base }
7989 \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
7990 }

```

```

7991     {
7992         \str_if_eq:eeTF { \l_@@_vpos_block_str } { b }
7993         {
7994             \@@_qpoint:n { row - #3 - base }
7995             \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
7996         }
7997         {
7998             \@@_qpoint:n { #1 - #2 - block }
7999             \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8000         }
8001     }
8002 }
8003 }
8004 \int_step_inline:nn { \l_@@_split_int }
8005 {
8006     \group_begin:

```

The counter `\l_@@_split_i_int` is only for the command `\@@_subcellcolor`.

```

8007     \int_set:Nn \l_@@_split_i_int { ##1 }
8008     \dim_set:Nn \col@sep
8009     { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
8010     \pgftransformshift
8011     {
8012         \pgfpoint
8013         {
8014             \l_tmpa_dim + ##1 \l_tmpb_dim -
8015             \str_case:on \l_@@_hpos_block_str
8016             {
8017                 l { \l_tmpb_dim + \col@sep }
8018                 c { 0.5 \l_tmpb_dim }
8019                 r { \col@sep }
8020             }
8021         }
8022         { \l_@@_tmpc_dim }
8023     }
8024     \pgfset { inner~sep = \c_zero_dim }
8025     \pgfnode
8026     { rectangle }
8027     {
8028         \str_if_eq:eeTF { \l_@@_vpos_block_str } { T }
8029         {
8030             \str_case:on \l_@@_hpos_block_str
8031             {
8032                 l { north~west }
8033                 c { north }
8034                 r { north~east }
8035             }
8036         }
8037         {
8038             \str_if_eq:eeTF { \l_@@_vpos_block_str } { B }
8039             {
8040                 \str_case:on \l_@@_hpos_block_str
8041                 {
8042                     l { south~west }
8043                     c { south }
8044                     r { south~east }
8045                 }
8046             }
8047             {
8048                 \bool_lazy_any:nTF
8049                 {
8050                     { \int_compare_p:nNn { #1 } = { #3 } }
8051                     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
8052                     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { b } }

```

```

8053         }
8054         {
8055             \str_case:on \l_@@_hpos_block_str
8056             {
8057                 l { base~west }
8058                 c { base }
8059                 r { base~east }
8060             }
8061         }
8062         {
8063             \str_case:on \l_@@_hpos_block_str
8064             {
8065                 l { west }
8066                 c { center }
8067                 r { east }
8068             }
8069         }
8070     }
8071 }
8072 }
8073 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8074 \group_end:
8075 }
8076 \endpgfpicture
8077 }

```

Now the case where there is no ampersand & in the content of the block.

```

8078 {
8079     \bool_if:NTF \l_@@_p_block_bool
8080     {

```

When the final user has used the key p, we have to compute the width.

```

8081     \pgfpicture
8082     \pgfrememberpicturepositiononpagetrue
8083     \pgf@relevantforpicturesizefalse
8084     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8085     {
8086         \@@_qpoint:n { col - #2 }
8087         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8088         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8089     }
8090     {
8091         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8092         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8093         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8094     }
8095     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8096 \endpgfpicture
8097 \hbox_set:Nn \l_@@_cell_box
8098 {
8099     \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8100     { \g_tmpb_dim }
8101     \str_case:on \l_@@_hpos_block_str
8102     { c \centering r \raggedleft l \raggedright j { } }
8103     #6
8104     \end { minipage }
8105 }
8106 }
8107 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
8108 \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.

```

8109 \pgfpicture

```

```

8110 \pgfrememberpicturepositiononpagetrue
8111 \pgf@relevantforpicturesizefalse
8112 \bool_lazy_any:nTF
8113 {
8114   { \str_if_empty_p:N \l_@@_vpos_block_str }
8115   { \str_if_eq_p:ee { \l_@@_vpos_block_str } { c } }
8116   { \str_if_eq_p:ee { \l_@@_vpos_block_str } { T } }
8117   { \str_if_eq_p:ee { \l_@@_vpos_block_str } { B } }
8118 }
8119 {

```

If we are in the “first column”, we must put the block as if it was with the key `r`.

```

8120 \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`.

```

8121 \bool_if:nT \g_@@_last_col_found_bool
8122 {
8123   \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
8124   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8125 }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8126 \tl_set:Ne \l_tmpa_tl
8127 {
8128   \str_case:on \l_@@_vpos_block_str
8129   {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8130   { } {
8131     \str_case:on \l_@@_hpos_block_str
8132     {
8133       c { center }
8134       l { west }
8135       r { east }
8136       j { center }
8137     }
8138   }
8139   c {
8140     \str_case:on \l_@@_hpos_block_str
8141     {
8142       c { center }
8143       l { west }
8144       r { east }
8145       j { center }
8146     }
8147   }
8148   T {
8149     \str_case:on \l_@@_hpos_block_str
8150     {
8151       c { north }
8152       l { north-west }
8153       r { north-east }
8154       j { north }
8155     }
8156   }
8157   B {
8158     \str_case:on \l_@@_hpos_block_str
8159     {
8160       c { south }
8161       l { south-west }
8162       r { south-east }

```

```

8165             j { south }
8166         }
8167
8168     }
8169 }
8170
8171 \pgftransformshift
8172 {
8173     \pgfpointanchor
8174     {
8175         \@@_env: - #1 - #2 - block
8176         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8177     }
8178     { \l_tmpa_tl }
8179 }
8180 \pgfset { inner~sep = \c_zero_dim }
8181 \pgfnode
8182 { rectangle }
8183 { \l_tmpa_tl }
8184 { \box_use_drop:N \l_@@_cell_box } { } { }
8185 }

```

End of the case when `\l_@@_vpos_block_str` is equal to c, T or B. Now, the other cases.

```

8186 {
8187     \pgfextracty \l_tmpa_dim
8188     {
8189         \@@_qpoint:n
8190         {
8191             row - \str_if_eq:eeTF { \l_@@_vpos_block_str } { b } { #3 } { #1 }
8192             - base
8193         }
8194     }
8195     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

8196 \pgfpointanchor
8197 {
8198     \@@_env: - #1 - #2 - block
8199     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8200 }
8201 {
8202     \str_case:on \l_@@_hpos_block_str
8203     {
8204         c { center }
8205         l { west }
8206         r { east }
8207         j { center }
8208     }
8209 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8210 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8211 \pgfset { inner~sep = \c_zero_dim }
8212 \pgfnode
8213 { rectangle }
8214 {
8215     \str_case:on \l_@@_hpos_block_str
8216     {
8217         c { base }
8218         l { base-west }
8219         r { base-east }
8220         j { base }
8221     }
8222 }

```

```

8223         { \box_use_drop:N \l_@@_cell_box } { } { }
8224     }
8225     \endpgfpicture
8226 }
8227 \group_end:
8228 }
8229 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }

```

The following command adds the value of `\l_@@_opacity_tl` (if not empty) to the specification of color set in `\l_@@_fill_tl` (the information of opacity is added in between square brackets before the color itself).

```

8230 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8231 {
8232     \tl_if_empty:NF \l_@@_opacity_tl
8233     {
8234         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8235             {
8236                 \tl_set:Ne \l_@@_fill_tl
8237                 {
8238                     [ opacity = \l_@@_opacity_tl ,
8239                     \tl_tail:o \l_@@_fill_tl
8240                 }
8241             }
8242             {
8243                 \tl_set:Ne \l_@@_fill_tl
8244                 { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8245             }
8246         }
8247     }

```

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).

```

8248 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8249 {
8250     \group_begin:
8251     \tl_clear:N \l_@@_draw_tl
8252     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8253     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8254     \pgfpicture
8255     \pgfrememberpicturepositiononpagetrue
8256     \pgf@relevantforpicturesizefalse
8257     \tl_if_empty:NF \l_@@_draw_tl
8258     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8259         \tl_if_eq:NnTF \l_@@_draw_tl { default }
8260         { \CT@arc@ }
8261         { \@@_color:o \l_@@_draw_tl }
8262     }
8263     \pgfsetcornersarced
8264     {
8265         \pgfpoint
8266         { \l_@@_rounded_corners_dim }
8267         { \l_@@_rounded_corners_dim }
8268     }
8269     \@@_cut_on_hyphen:w #2 \q_stop
8270     \int_compare:nNf { \l_tmpa_tl } > { \c@iRow }
8271     {
8272         \int_compare:nNf { \l_tmpb_tl } > { \c@jCol }
8273         {

```



```

8274 \@@_qpoint:n { row - \l_tmpa_tl }
8275 \dim_set_eq:NN \l_tmpb_dim \pgf@y
8276 \@@_qpoint:n { col - \l_tmpb_tl }
8277 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8278 \@@_cut_on_hyphen:w #3 \q_stop
8279 \int_compare:nNnT { \l_tmpa_tl } > { \c@iRow }
8280 { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8281 \int_compare:nNnT { \l_tmpb_tl } > { \c@jCol }
8282 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8283 \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8284 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8285 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8286 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8287 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8288 \pgfpathrectanglecorners
8289 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8290 { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8291 \dim_compare:nNnTF { \l_@@_rounded_corners_dim } = { \c_zero_dim }
8292 { \pgfusepathqstroke }
8293 { \pgfusepath { stroke } }
8294 }
8295 }
8296 \endpgfpicture
8297 \group_end:
8298 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8299 \keys_define:nn { nicematrix / BlockStroke }
8300 {
8301   color .tl_set:N = \l_@@_draw_tl ,
8302   draw .code:n =
8303     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8304   draw .default:n = default ,
8305   line-width .dim_set:N = \l_@@_line_width_dim ,
8306   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8307   rounded-corners .default:n = 4 pt
8308 }

```

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).

```

8309 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8310 {
8311   \group_begin:
8312   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8313   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8314   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8315   \@@_cut_on_hyphen:w #2 \q_stop
8316   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8317   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8318   \@@_cut_on_hyphen:w #3 \q_stop
8319   \tl_set:Nc \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8320   \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8321   \int_step_inline:nnn { \l_@@_tmpd_tl } { \l_tmpb_tl }
8322   {
8323     \use:e
8324     {
8325       \@@_vline:n
8326       {
8327         position = ##1 ,
8328         start = \l_@@_tmpc_tl ,
8329         end = \int_eval:n { \l_tmpa_tl - 1 } ,
8330         total-width = \dim_use:N \l_@@_line_width_dim

```

```

8331     }
8332   }
8333 }
8334 \group_end:
8335 }
8336 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8337 {
8338   \group_begin:
8339   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8340   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8341   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8342   \@@_cut_on_hyphen:w #2 \q_stop
8343   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8344   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8345   \@@_cut_on_hyphen:w #3 \q_stop
8346   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8347   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8348   \int_step_inline:nnn { \l_@@_tmpc_tl } { \l_tmpa_tl }
8349   {
8350     \use:e
8351     {
8352       \@@_hline:n
8353       {
8354         position = ##1 ,
8355         start = \l_@@_tmpd_tl ,
8356         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8357         total-width = \dim_use:N \l_@@_line_width_dim
8358       }
8359     }
8360   }
8361   \group_end:
8362 }

```

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).

```

8363 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8364 {
8365   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8366   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8367   \dim_compare:nNnTF { \l_@@_rounded_corners_dim } > { \c_zero_dim }
8368   { \@@_error:n { borders~forbidden } }
8369   {
8370     \tl_clear_new:N \l_@@_borders_tikz_tl
8371     \keys_set:no
8372     { nicematrix / OnlyForTikzInBorders }
8373     \l_@@_borders_clist
8374     \@@_cut_on_hyphen:w #2 \q_stop
8375     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8376     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8377     \@@_cut_on_hyphen:w #3 \q_stop
8378     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8379     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8380     \@@_stroke_borders_block_i:
8381   }
8382 }
8383 \hook_gput_code:nnn { begindocument } { . }
8384 {
8385   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8386   {
8387     \c_@@_pgfortikzpicture_tl
8388     \@@_stroke_borders_block_ii:

```

```

8389         \c_@@_endpgfortikzpicture_tl
8390     }
8391 }

8392 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8393 {
8394     \pgfrememberpicturepositiononpagetrue
8395     \pgf@relevantforpicturesizefalse
8396     \CT@arc@
8397     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8398     \clist_if_in:NnT \l_@@_borders_clist { right }
8399     { \@@_stroke_vertical:n \l_tmpb_tl }
8400     \clist_if_in:NnT \l_@@_borders_clist { left }
8401     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8402     \clist_if_in:NnT \l_@@_borders_clist { bottom }
8403     { \@@_stroke_horizontal:n \l_tmpa_tl }
8404     \clist_if_in:NnT \l_@@_borders_clist { top }
8405     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8406 }

8407 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8408 {
8409     tikz .code:n =
8410         \cs_if_exist:NTF \tikzpicture
8411         { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8412         { \@@_error:n { tikz~in~borders~without~tikz } } ,
8413     tikz .value_required:n = true ,
8414     top .code:n = ,
8415     bottom .code:n = ,
8416     left .code:n = ,
8417     right .code:n = ,
8418     unknown .code:n = \@@_error:n { bad~border }
8419 }

```

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).

```

8420 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8421 {
8422     \@@_qpoint:n \l_@@_tmpc_tl
8423     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8424     \@@_qpoint:n \l_tmpa_tl
8425     \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8426     \@@_qpoint:n { #1 }
8427     \tl_if_empty:NTF \l_@@_borders_tikz_tl
8428     {
8429         \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8430         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8431         \pgfusepath{stroke}
8432     }
8433     {
8434         \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8435         ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8436     }
8437 }

```

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).

```

8438 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8439 {
8440     \@@_qpoint:n \l_@@_tmpd_tl
8441     \clist_if_in:NnTF \l_@@_borders_clist { left }
8442     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8443     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8444     \@@_qpoint:n \l_tmpb_tl

```

```

8445 \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8446 \@@_qpoint:n { #1 }
8447 \tl_if_empty:NTF \l_@@_borders_tikz_tl
8448 {
8449   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8450   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8451   \pgfusepathqstroke
8452 }
8453 {
8454   \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8455   ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8456 }
8457 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

8458 \keys_define:nn { nicematrix / BlockBorders }
8459 {
8460   borders .clist_set:N = \l_@@_borders_clist ,
8461   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8462   rounded-corners .default:n = 4 pt ,
8463   line-width .dim_set:N = \l_@@_line_width_dim
8464 }

```

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.

```

8465 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8466 {
8467   \begin { tikzpicture }
8468   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because **#5** is a list of lists.

```

8469   \clist_map_inline:nn { #1 }
8470   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8471   \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8472   \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8473   (
8474     [
8475       xshift = \dim_use:N \l_@@_offset_dim ,
8476       yshift = - \dim_use:N \l_@@_offset_dim
8477     ]
8478     #2 -| #3
8479   )
8480   rectangle
8481   (
8482     [
8483       xshift = - \dim_use:N \l_@@_offset_dim ,
8484       yshift = \dim_use:N \l_@@_offset_dim
8485     ]
8486     \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8487   ) ;
8488   }
8489   \end { tikzpicture }
8490 }
8491 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

```

```

8492 \keys_define:nn { nicematrix / SpecialOffset }
8493 { 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.

```

8494 \cs_new_protected:Npn \@@_NullBlock:
8495 { \@@_collect_options:n { \@@_NullBlock_i: } }
8496 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8497 { }

```

The following command will be linked to `\cellcolor` in the sub-cells of a block which contains ampersands (&). Of course, `&-in-blocks` must be in force.

```

8498 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
8499 {
8500   \tl_gput_right:Ne \g_@@_pre_code_before_tl
8501   {

```

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`).

```

8502     \@@_subcellcolor:nnnnnnn
8503     {
8504       \tl_if_blank:nTF { #1 }
8505       { { \exp_not:n { #2 } } }
8506       { [ #1 ] { \exp_not:n { #2 } } }
8507     }
8508     { \int_use:N \l_@@_first_row_int } % first row of the block
8509     { \int_use:N \l_@@_first_col_int } % first column of the block
8510     { \int_use:N \l_@@_last_row_int } % last row of the block
8511     { \int_use:N \l_@@_last_col_int } % last column of the block
8512     { \int_use:N \l_@@_split_int }
8513     { \int_use:N \l_@@_split_i_int }
8514   }
8515   \ignorespaces
8516 }

8517 \cs_new_protected:Npn \@@_subcellcolor:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8518 {
8519   \@@_color_opacity: #1
8520   \pgfpicture
8521   \pgf@relevantforpicturesizefalse
8522   \@@_qpoint:n { col - #3 }
8523   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8524   \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
8525   \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
8526   \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
8527   \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
8528   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8529   \@@_qpoint:n { row - #2 }
8530   \pgfpathrectanglecorners
8531   { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } { \l_@@_tmpc_dim } }
8532   { \pgfpoint { \l_tmpb_dim } { \pgf@y } }
8533   \pgfusepathqfill
8534   \endpgfpicture
8535 }

```

27 How to draw the dotted lines transparently

```

8536 \cs_set_protected:Npn \@@_renew_matrix:
8537 {

```

```

8538 \RenewDocumentEnvironment { pmatrix } { }
8539 { \pNiceMatrix }
8540 { \endpNiceMatrix }
8541 \RenewDocumentEnvironment { vmatrix } { }
8542 { \vNiceMatrix }
8543 { \endvNiceMatrix }
8544 \RenewDocumentEnvironment { Vmatrix } { }
8545 { \VNiceMatrix }
8546 { \endVNiceMatrix }
8547 \RenewDocumentEnvironment { bmatrix } { }
8548 { \bNiceMatrix }
8549 { \endbNiceMatrix }
8550 \RenewDocumentEnvironment { Bmatrix } { }
8551 { \BNiceMatrix }
8552 { \endBNiceMatrix }
8553 }

```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8554 \keys_define:nn { nicematrix / Auto }
8555 {
8556   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8557   columns-type .value_required:n = true ,
8558   l .meta:n = { columns-type = l } ,
8559   r .meta:n = { columns-type = r } ,
8560   c .meta:n = { columns-type = c } ,
8561   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8562   delimiters / color .value_required:n = true ,
8563   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8564   delimiters / max-width .default:n = true ,
8565   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8566   delimiters .value_required:n = true ,
8567   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8568   rounded-corners .default:n = 4 pt
8569 }
8570 \NewDocumentCommand \AutoNiceMatrixWithDelims
8571 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8572 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
8573 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8574 {

```

The group is for the protection of the keys.

```

8575 \group_begin:
8576 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8577 \use:e
8578 {
8579   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8580   { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8581   [ \exp_not:o \l_tmpa_tl ]
8582 }
8583 \int_if_zero:nT { \l_@@_first_row_int }
8584 {
8585   \int_if_zero:nT { \l_@@_first_col_int } { & }
8586   \prg_replicate:nn { #4 - 1 } { & }
8587   \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8588 }
8589 \prg_replicate:nn { #3 }
8590 {
8591   \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`).

```

8592     \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8593     \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8594   }
8595   \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
8596   {
8597     \int_if_zero:nT { \l_@@_first_col_int } { & }
8598     \prg_replicate:nn { #4 - 1 } { & }
8599     \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8600   }
8601   \end { NiceArrayWithDelims }
8602   \group_end:
8603 }

8604 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
8605 {
8606   \cs_set_protected:cpn { #1 AutoNiceMatrix }
8607   {
8608     \bool_gset_true:N \g_@@_delims_bool
8609     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8610     \AutoNiceMatrixWithDelims { #2 } { #3 }
8611   }
8612 }

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8613 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8614 {
8615   \group_begin:
8616   \bool_gset_false:N \g_@@_delims_bool
8617   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8618   \group_end:
8619 }

```

29 The redefinition of the command `\dotfill`

```

8620 \cs_set_eq:NN \@@_old_dotfill: \dotfill
8621 \cs_new_protected:Npn \@@_dotfill:
8622 {

```

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}`).

```

8623   \@@_old_dotfill:
8624   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8625 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

8626 \cs_new_protected:Npn \@@_dotfill_i:
8627 {
8628   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = { \c_zero_dim }
8629   { \@@_old_dotfill: }
8630 }

```

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.

```

8631 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8632 {
8633   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8634   {
8635     \@@_actually_diagbox:nnnnnn
8636     { \int_use:N \c@iRow }
8637     { \int_use:N \c@jCol }
8638     { \int_use:N \c@iRow }
8639     { \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.

```

8640     { \g_@@_row_style_tl \exp_not:n { #1 } }
8641     { \g_@@_row_style_tl \exp_not:n { #2 } }
8642   }

```

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`.

```

8643   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8644   {
8645     { \int_use:N \c@iRow }
8646     { \int_use:N \c@jCol }
8647     { \int_use:N \c@iRow }
8648     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8649     { }
8650   }
8651 }

```

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.

```

8652 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8653 {
8654   \pgfpicture
8655   \pgf@relevantforpicturesizefalse
8656   \pgfrememberpicturepositiononpagetrue
8657   \@@_qpoint:n { row - #1 }
8658   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8659   \@@_qpoint:n { col - #2 }
8660   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8661   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8662   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8663   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8664   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8665   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8666   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8667   {

```

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.

```

8668   \CT@arc@
8669   \pgfsetroundcap

```



```

8670     \pgfusepathqstroke
8671   }
8672   \pgfset { inner~sep = 1 pt }
8673   \pgfscope
8674   \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8675   \pgfnode { rectangle } { south~west }
8676   {
8677     \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8678     \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8679   \end { minipage }
8680   }
8681   { }
8682   { }
8683 \endpgfscope
8684 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8685 \pgfnode { rectangle } { north~east }
8686 {
8687   \begin { minipage } { 20 cm }
8688   \raggedleft
8689   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8690   \end { minipage }
8691 }
8692 { }
8693 { }
8694 \endpgfpicture
8695 }

```

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`.

```

8696 \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 `\`.

```

8697 \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`.

```

8698 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8699 {
8700   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8701   \@@_CodeAfter_iv:n
8702 }

```

We catch the argument of the command `\end` (in #1).

```

8703 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8704 {

```

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.

```

8705   \str_if_eq:eeTF { \currenvir } { #1 }
8706   { \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`.

```

8707     {
8708         \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8709         \@@_CodeAfter_ii:n
8710     }
8711 }

```

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.

```

8712 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8713 {
8714     \pgfpicture
8715     \pgfrememberpicturepositiononpagetrue
8716     \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.

```

8717     \@@_qpoint:n { row - 1 }
8718     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8719     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8720     \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).

```

8721     \bool_if:nTF { #3 }
8722     { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8723     { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8724     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8725     {
8726         \cs_if_exist:cT
8727         { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8728         {
8729             \pgfpointanchor
8730             { \@@_env: - ##1 - #2 }
8731             { \bool_if:nTF { #3 } { west } { east } }
8732             \dim_set:Nn \l_tmpa_dim
8733             {
8734                 \bool_if:nTF { #3 }
8735                 { \dim_min:nn }
8736                 { \dim_max:nn }
8737                 \l_tmpa_dim
8738                 { \pgf@x }
8739             }
8740         }
8741     }

```

Now we can put the delimiter with a node of PGF.

```

8742 \pgfset { inner~sep = \c_zero_dim }
8743 \dim_zero:N \nulldelimiterspace
8744 \pgftransformshift
8745 {
8746   \pgfpoint
8747   { \l_tmpa_dim }
8748   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8749 }
8750 \pgfnode
8751 { rectangle }
8752 { \bool_if:nTF { #3 } { east } { west } }
8753 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8754 \nullfont
8755 $ % $
8756 \@@_color:o \l_@@_delimiters_color_tl
8757 \bool_if:nTF { #3 } { \left #1 } { \left . }
8758 \vcenter
8759 {
8760   \nullfont
8761   \hrule \@height
8762   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8763   \@depth \c_zero_dim
8764   \@width \c_zero_dim
8765 }
8766 \bool_if:nTF { #3 } { \right . } { \right #1 }
8767 $ % $
8768 }
8769 { }
8770 { }
8771 \endpgfpicture
8772 }

```

33 The command \SubMatrix

```

8773 \keys_define:nn { nicematrix / sub-matrix }
8774 {
8775   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8776   extra-height .value_required:n = true ,
8777   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8778   left-xshift .value_required:n = true ,
8779   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8780   right-xshift .value_required:n = true ,
8781   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8782   xshift .value_required:n = true ,
8783   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8784   delimiters / color .value_required:n = true ,
8785   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8786   slim .default:n = true ,
8787   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8788   hlines .default:n = all ,
8789   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8790   vlines .default:n = all ,
8791   hvlines .meta:n = { hlines, vlines } ,
8792   hvlines .value_forbidden:n = true
8793 }
8794 \keys_define:nn { nicematrix }
8795 {
8796   SubMatrix .inherit:n = nicematrix / sub-matrix ,

```

```

8797 NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8798 pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8799 NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8800 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8801 \keys_define:nn { nicematrix / SubMatrix }
8802 {
8803   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8804   delimiters / color .value_required:n = true ,
8805   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8806   hlines .default:n = all ,
8807   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8808   vlines .default:n = all ,
8809   hvlines .meta:n = { hlines, vlines } ,
8810   hvlines .value_forbidden:n = true ,
8811   name .code:n =
8812     \tl_if_empty:nTF { #1 }
8813     { \@@_error:n { Invalid-name } }
8814     {
8815       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8816       {
8817         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8818         { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
8819         {
8820           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8821           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8822         }
8823       }
8824       { \@@_error:n { Invalid-name } }
8825     } ,
8826   name .value_required:n = true ,
8827   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8828   rules .value_required:n = true ,
8829   code .tl_set:N = \l_@@_code_tl ,
8830   code .value_required:n = true ,
8831   unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8832 }

8833 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8834 {
8835   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8836   {
8837     \SubMatrix { #1 } { #2 } { #3 } { #4 }
8838     [
8839       delimiters / color = \l_@@_delimiters_color_tl ,
8840       hlines = \l_@@_submatrix_hlines_clist ,
8841       vlines = \l_@@_submatrix_vlines_clist ,
8842       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8843       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8844       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8845       slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8846       #5
8847     ]
8848   }
8849   \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8850   \ignorespaces
8851 }

8852 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8853 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8854 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

```

```

8855 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8856 {
8857   \seq_gput_right:Ne \g_@@_submatrix_seq
8858   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8859     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8860     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8861     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8862     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8863   }
8864 }

```

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).

```

8865 \NewDocumentCommand \@@_compute_i_j:nn
8866 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8867 { \@@_compute_i_j:nnnn #1 #2 }

```

```

8868 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8869 {
8870   \def \l_@@_first_i_tl { #1 }
8871   \def \l_@@_first_j_tl { #2 }
8872   \def \l_@@_last_i_tl { #3 }
8873   \def \l_@@_last_j_tl { #4 }
8874   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8875     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8876   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8877     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8878   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8879     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8880   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8881     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8882 }

```

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`.

```

8883 \hook_gput_code:nnn { begindocument } { . }
8884 {
8885   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
8886   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
8887     { \@@_sub_matrix:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
8888 }
8889 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
8890 {
8891   \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

8892 \@@_compute_i_j:nn { #2 } { #3 }
8893 \int_compare:nNt { \l_@@_first_i_tl } = { \l_@@_last_i_tl }
8894 { \def \arraystretch { 1 } }
8895 \bool_lazy_or:nnTF
8896 { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
8897 { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
8898 { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8899 {
8900 \str_clear_new:N \l_@@_submatrix_name_str
8901 \keys_set:nn { nicematrix / SubMatrix } { #5 }
8902 \pgfpicture
8903 \pgfrememberpicturepositiononpagetrue
8904 \pgf@relevantforpicturesizefalse
8905 \pgfset { inner~sep = \c_zero_dim }
8906 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8907 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int_step_inline:nnn is provided by curryfication.

```

8908 \bool_if:NTF \l_@@_submatrix_slim_bool
8909 { \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl } }
8910 { \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int } }
8911 {
8912 \cs_if_exist:cT
8913 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8914 {
8915 \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8916 \dim_compare:nNt { \pgf@x } < { \l_@@_x_initial_dim }
8917 { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8918 }
8919 \cs_if_exist:cT
8920 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8921 {
8922 \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8923 \dim_compare:nNt { \pgf@x } > { \l_@@_x_final_dim }
8924 { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8925 }
8926 }
8927 \dim_compare:nNtTF { \l_@@_x_initial_dim } = { \c_max_dim }
8928 { \@@_error:nn { Impossible~delimiter } { left } }
8929 {
8930 \dim_compare:nNtTF { \l_@@_x_final_dim } = { - \c_max_dim }
8931 { \@@_error:nn { Impossible~delimiter } { right } }
8932 { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8933 }
8934 \endpgfpicture
8935 }
8936 \group_end:
8937 \ignorespaces
8938 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8939 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8940 {
8941 \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8942 \dim_set:Nn \l_@@_y_initial_dim
8943 {
8944 \fp_to_dim:n
8945 {
8946 \pgf@y
8947 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8948 }
8949 }

```

```

8950 \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8951 \dim_set:Nn \l_@@_y_final_dim
8952 { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8953 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
8954 {
8955   \cs_if_exist:cT
8956   { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8957   {
8958     \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8959     \dim_set:Nn \l_@@_y_initial_dim
8960     { \dim_max:nn { \l_@@_y_initial_dim } { \pgf@y } }
8961   }
8962   \cs_if_exist:cT
8963   { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8964   {
8965     \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8966     \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
8967     { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8968   }
8969 }
8970 \dim_set:Nn \l_tmpa_dim
8971 {
8972   \l_@@_y_initial_dim - \l_@@_y_final_dim +
8973   \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8974 }
8975 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8976 \group_begin:
8977 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8978 \@@_set_CTarc:o \l_@@_rules_color_tl
8979 \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`.

```

8980 \seq_map_inline:Nn \g_@@_cols_vlism_seq
8981 {
8982   \int_compare:nNnT { \l_@@_first_j_tl } < { ##1 }
8983   {
8984     \int_compare:nNnT
8985     { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8986     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8987 \@@_qpoint:n { col - ##1 }
8988 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8989 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8990 \pgfusepathqstroke
8991 }
8992 }
8993 }

```

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.

```

8994 \str_if_eq:eeTF { \l_@@_submatrix_vlines_clist } { all }
8995 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8996 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8997 {
8998   \bool_lazy_and:nnTF
8999   { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
9000   {
9001     \int_compare_p:nNn

```

```

9002         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
9003     {
9004         \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
9005         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9006         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9007         \pgfusepathqstroke
9008     }
9009     { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
9010 }

```

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.

```

9011 \str_if_eq:eeTF { \l_@@_submatrix_hlines_clist } { all }
9012 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
9013 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
9014 {
9015     \bool_lazy_and:nnTF
9016     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
9017     {
9018         \int_compare_p:nNn
9019         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
9020     {
9021         \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

9022     \group_begin:
We compute in \l_tmpa_dim the x-value of the left end of the rule.
9023     \dim_set:Nn \l_tmpa_dim
9024     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9025     \str_case:nn { #1 }
9026     {
9027         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9028         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
9029         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9030     }
9031     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the *x*-value of the right end of the rule.

```

9032     \dim_set:Nn \l_tmpb_dim
9033     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9034     \str_case:nn { #2 }
9035     {
9036         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9037         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
9038         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9039     }
9040     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9041     \pgfusepathqstroke
9042     \group_end:
9043 }
9044 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
9045 }

```

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).

```

9046 \str_if_empty:NF \l_@@_submatrix_name_str
9047 {
9048     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
9049     \l_@@_x_initial_dim \l_@@_y_initial_dim
9050     \l_@@_x_final_dim \l_@@_y_final_dim
9051 }
9052 \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`.

```

9053   \begin { pgfscope }
9054   \pgftransformshift
9055   {
9056     \pgfpoint
9057     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9058     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9059   }
9060   \str_if_empty:NTF \l_@@_submatrix_name_str
9061   { \@@_node_left:nn #1 { } }
9062   { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9063   \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9064   \pgftransformshift
9065   {
9066     \pgfpoint
9067     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9068     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9069   }
9070   \str_if_empty:NTF \l_@@_submatrix_name_str
9071   { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9072   {
9073     \@@_node_right:nnnn #2
9074     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9075   }

```

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`.

```

9076   \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9077   \flag_clear_new:N \l_@@_code_flag
9078   \l_@@_code_tl
9079 }

```

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`.

```

9080 \cs_set_eq:NN \@@_old_pgfpntanchor: \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by currying.

```

9081 \cs_new:Npn \@@_pgfpointanchor:n #1
9082 { \exp_args:Ne \@@_old_pgfpntanchor: { \@@_pgfpointanchor_i:n { #1 } } }

```

First, we must detect whether the argument is of the form `\tikz@pp@name{...}` (the command `\tikz@pp@name` is a command of TikZ that adds the prefix and the suffix of the name. If the name refers to a TikZ node which does not exist, there isn't the wrapper `\tikz@pp@name`).

```

9083 \cs_new:Npn \@@_pgfpointanchor_i:n #1
9084 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }

```

```

9085 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9086 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9087 \str_if_empty:nTF { #1 }

```

First, when the name of the name begins with `\tikz@pp@name`.

```

9088 { \@@_pgfpointanchor_iv:w #2 }

```

And now, when there is no `\tikz@pp@name`.

```

9089 { \@@_pgfpointanchor_ii:n { #1 } }
9090 }

```

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`).

```

9091 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
9092 { \@@_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`.

```

9093 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

```

```

9094 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
9095 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9096 \str_if_empty:nTF { #2 }

```

First the case where the argument does *not* contain an hyphen.

```

9097 { \@@_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`).

```

9098 { \@@_pgfpointanchor_iii:w { #1 } #2 }
9099 }

```

The following function is for the case when the name contains an hyphen.

```

9100 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9101 {

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9102 \@@_env:
9103 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9104 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9105 }

```

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.

```

9106 \tl_const:Nn \c_@@_integers_alist_tl
9107 {
9108 { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9109 { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9110 { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9111 { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9112 }

```

```

9113 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9114 {

```

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`.

```

9115 \str_case:nVTF { #1 } \c_@@_integers_alist_tl
9116 {
9117   \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `@@_env`: “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9118   @@_env: -
9119   \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9120     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9121     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9122   }
9123   {
9124     \str_if_eq:eeTF { #1 } { last }
9125     {
9126       \flag_raise:N \l_@@_code_flag
9127       @@_env: -
9128       \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9129         { \int_eval:n { \l_@@_last_i_tl + 1 } }
9130         { \int_eval:n { \l_@@_last_j_tl + 1 } }
9131     }
9132     { #1 }
9133   }
9134 }

```

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`).

```

9135 \cs_new_protected:Npn @@_node_left:nn #1 #2
9136 {
9137   \pgfnode
9138     { rectangle }
9139     { east }
9140     {
9141       \nullfont
9142       $ % $
9143       @@_color:o \l_@@_delimiters_color_tl
9144       \left #1
9145       \vcenter
9146       {
9147         \nullfont
9148         \hrule \@height \l_tmpa_dim
9149           \@depth \c_zero_dim
9150           \@width \c_zero_dim
9151       }
9152       \right .
9153       $ % $
9154     }
9155     { #2 }
9156     { }
9157 }

```

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.

```

9158 \cs_new_protected:Npn @@_node_right:nnnn #1 #2 #3 #4

```

```

9159 {
9160   \pgfnode
9161   { rectangle }
9162   { west }
9163   {
9164     \nullfont
9165     $ % $
9166     \colorlet { current-color } { . }
9167     \@@_color:o \l_@@_delimiters_color_tl
9168     \left .
9169     \vcenter
9170     {
9171       \nullfont
9172       \hrule \@height \l_tmpa_dim
9173       \@depth \c_zero_dim
9174       \@width \c_zero_dim
9175     }
9176     \right #1
9177     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9178     ^ { \color { current-color } \smash { #4 } }
9179     $ % $
9180   }
9181   { #2 }
9182   { }
9183 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9184 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9185 {
9186   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9187   \ignorespaces
9188 }
9189 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9190 {
9191   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9192   \ignorespaces
9193 }
9194 \keys_define:nn { nicematrix / Brace }
9195 {
9196   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9197   left-shorten .default:n = true ,
9198   left-shorten .value_forbidden:n = true ,
9199   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9200   right-shorten .default:n = true ,
9201   right-shorten .value_forbidden:n = true ,
9202   shorten .meta:n = { left-shorten , right-shorten } ,
9203   shorten .value_forbidden:n = true ,
9204   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9205   yshift .value_required:n = true ,
9206   yshift .initial:n = \c_zero_dim ,
9207   color .tl_set:N = \l_tmpa_tl ,
9208   color .value_required:n = true ,
9209   unknown .code:n =
9210     \@@_unknown_key:nn
9211     { nicematrix / Brace }

```

```

9212     { Unknown-key-for-Brace }
9213 }

```

#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`.

```

9214 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9215 {
9216   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9217   \@@_compute_i_j:nn { #1 } { #2 }
9218   \bool_lazy_or:nnTF
9219     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9220     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9221   {
9222     \str_if_eq:eeTF { #5 } { under }
9223     { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9224     { \@@_error:nn { Construct-too-large } { \OverBrace } }
9225   }
9226   {
9227     \tl_clear:N \l_tmpa_tl
9228     \keys_set:nn { nicematrix / Brace } { #4 }
9229     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9230     \pgfpicture
9231     \pgfrememberpicturepositiononpagetrue
9232     \pgf@relevantforpicturesizefalse
9233     \bool_if:NT \l_@@_brace_left_shorten_bool
9234     {
9235       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9236       \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9237       {
9238         \cs_if_exist:cT
9239           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9240           {
9241             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9242
9243             \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9244               { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9245           }
9246       }
9247     }
9248     \bool_lazy_or:nnT
9249       { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9250       { \dim_compare_p:nNn { \l_@@_x_initial_dim } = { \c_max_dim } }
9251     {
9252       \@@_qpoint:n { col - \l_@@_first_j_tl }
9253       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9254     }
9255     \bool_if:NT \l_@@_brace_right_shorten_bool
9256     {
9257       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9258       \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9259       {
9260         \cs_if_exist:cT
9261           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9262           {
9263             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9264             \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
9265               { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9266           }
9267       }
9268     }
9269     \bool_lazy_or:nnT

```

```

9270     { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9271     { \dim_compare_p:nNn { \l_@@_x_final_dim } = { - \c_max_dim } }
9272     {
9273       \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9274       \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9275     }
9276     \pgfset { inner~sep = \c_zero_dim }
9277     \str_if_eq:eeTF { #5 } { under }
9278     { \@@_underbrace_i:n { #3 } }
9279     { \@@_overbrace_i:n { #3 } }
9280     \endpgfpicture
9281   }
9282   \group_end:
9283 }

```

The argument is the text to put above the brace.

```

9284 \cs_new_protected:Npn \@@_overbrace_i:n #1
9285 {
9286   \@@_qpoint:n { row - \l_@@_first_i_tl }
9287   \pgftransformshift
9288   {
9289     \pgfpoint
9290     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9291     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9292   }
9293   \pgfnode
9294   { rectangle }
9295   { south }
9296   {
9297     \vtop
9298     {
9299       \group_begin:
9300       \everycr { }
9301       \halign
9302       {
9303         \hfil ## \hfil \crcr
9304         \bool_if:NTF \l_@@_tabular_bool
9305         { \begin { tabular } { c } #1 \end { tabular } }
9306         { $ \begin { array } { c } #1 \end { array } $ }
9307         \cr
9308         $ % $
9309         \overbrace
9310         {
9311           \hbox_to_wd:nn
9312           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9313           { }
9314         }
9315         $ % $
9316         \cr
9317       }
9318       \group_end:
9319     }
9320   }
9321   { }
9322   { }
9323 }

```

The argument is the text to put under the brace.

```

9324 \cs_new_protected:Npn \@@_underbrace_i:n #1
9325 {
9326   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9327   \pgftransformshift
9328   {

```

```

9329     \pgfpoint
9330     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9331     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9332 }
9333 \pgfnode
9334 { rectangle }
9335 { north }
9336 {
9337     \group_begin:
9338     \everycr { }
9339     \vbox
9340     {
9341         \halign
9342         {
9343             \hfil ## \hfil \crcr
9344             $ % $
9345             \underbrace
9346             {
9347                 \hbox_to_wd:nn
9348                 { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9349                 { }
9350             }
9351             $ % $
9352             \cr
9353             \bool_if:NTF \l_@@_tabular_bool
9354             { \begin { tabular } { c } #1 \end { tabular } }
9355             { $ \begin { array } { c } #1 \end { array } $ }
9356             \cr
9357         }
9358     }
9359     \group_end:
9360 }
9361 { }
9362 { }
9363 }

```

35 The commands HBrace et VBrace

The TikZ style `nicematrix/brace` is a TikZ style used to draw the braces created by `\Hbrace` and `\Vbrace`.

We can't load that definition right away because of course, maybe the final user has not yet loaded TikZ (`\Hbrace` and `\Vbrace` are available only when TikZ is loaded and also its library `decorations.pathreplacing`).

```

9364 \AddToHook { package / tikz / after }
9365 {
9366     \tikzset
9367     {
9368         nicematrix / brace / .style =
9369         {
9370             decoration = { brace , raise = -0.15 em } ,
9371             decorate ,
9372         } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9373         nicematrix / mirrored-brace / .style =
9374         {
9375             nicematrix / brace ,
9376             decoration = mirror ,

```

```

9377     }
9378   }
9379 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9380 \keys_define:nn { nicematrix / Hbrace }
9381 {
9382   color .code:n = ,
9383   horizontal-label .code:n = ,
9384   horizontal-labels .code:n = ,
9385   shorten .code:n = ,
9386   shorten-start .code:n = ,
9387   shorten-end .code:n = ,
9388   unknown .code:n = \@@_fatal:n { Unknown-key-for-Hbrace }
9389 }

```

Here we need an “fully expandable” command.

```

9390 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9391 {
9392   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9393     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9394     { \@@_error:nn { Hbrace-not-allowed } { \Hbrace } }
9395 }

```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

9396 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9397 {
9398   \int_compare:nNnTF { \c@iRow } < { 2 }
9399   {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9400     \str_if_eq:nnTF { #2 } { * }
9401     {
9402       \bool_set_true:N \l_@@_nullify_dots_bool
9403       \Ldots
9404       [
9405         line-style = nicematrix / brace ,
9406         #1 ,
9407         up =
9408         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9409       ]
9410     }
9411     {
9412       \Hdotsfor
9413       [
9414         line-style = nicematrix / brace ,
9415         #1 ,
9416         up =
9417         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9418       ]
9419       { #2 }
9420     }
9421   }
9422   {
9423     \str_if_eq:nnTF { #2 } { * }
9424     {
9425       \bool_set_true:N \l_@@_nullify_dots_bool
9426       \Ldots
9427       [
9428         line-style = nicematrix / mirrored-brace ,

```



```

9429         #1 ,
9430         down =
9431         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9432     ]
9433 }
9434 {
9435     \Hdotsfor
9436     [
9437         line-style = nicematrix / mirrored-brace ,
9438         #1 ,
9439         down =
9440         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9441     ]
9442     { #2 }
9443 }
9444 }
9445 \keys_set:nn { nicematrix / Hbrace } { #1 }
9446 }

9447 \NewDocumentCommand { \@@_Vbrace } { 0 { } m m }
9448 {
9449     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9450     { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9451     { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9452 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

9453 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
9454 {
9455     \int_compare:nNnTF { \c@jCol } < { 2 }
9456     {
9457         \str_if_eq:nnTF { #2 } { * }
9458         {
9459             \bool_set_true:N \l_@@_nullify_dots_bool
9460             \Vdots
9461             [
9462                 Vbrace ,
9463                 line-style = nicematrix / mirrored-brace ,
9464                 #1 ,
9465                 down =
9466                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9467             ]
9468         }
9469         {
9470             \Vdotsfor
9471             [
9472                 Vbrace ,
9473                 line-style = nicematrix / mirrored-brace ,
9474                 #1 ,
9475                 down =
9476                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9477             ]
9478             { #2 }
9479         }
9480     }
9481     {
9482         \str_if_eq:nnTF { #2 } { * }
9483         {
9484             \bool_set_true:N \l_@@_nullify_dots_bool
9485             \Vdots
9486             [
9487                 Vbrace ,

```

```

9488         line-style = nicematrix / brace ,
9489         #1 ,
9490         up =
9491         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9492     ]
9493 }
9494 {
9495     \Vdotsfor
9496     [
9497         Vbrace ,
9498         line-style = nicematrix / brace ,
9499         #1 ,
9500         up =
9501         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9502     ]
9503     { #2 }
9504 }
9505 }
9506 \keys_set:nn { nicematrix / Hbrace } { #1 }
9507 }

```

36 The command TikzEveryCell

```

9508 \bool_new:N \l_@@_not_empty_bool
9509 \bool_new:N \l_@@_empty_bool
9510
9511 \keys_define:nn { nicematrix / TikzEveryCell }
9512 {
9513     not-empty .code:n =
9514         \bool_lazy_or:nnTF
9515         { \l_@@_in_code_after_bool }
9516         { \g_@@_create_cell_nodes_bool }
9517         { \bool_set_true:N \l_@@_not_empty_bool }
9518         { \@@_error:n { detection-of-empty-cells } } ,
9519     not-empty .value_forbidden:n = true ,
9520     empty .code:n =
9521         \bool_lazy_or:nnTF
9522         { \l_@@_in_code_after_bool }
9523         { \g_@@_create_cell_nodes_bool }
9524         { \bool_set_true:N \l_@@_empty_bool }
9525         { \@@_error:n { detection-of-empty-cells } } ,
9526     empty .value_forbidden:n = true ,
9527     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
9528 }
9529
9530
9531 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9532 {
9533     \IfPackageLoadedTF { tikz }
9534     {
9535         \group_begin:
9536         \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.

```

9537     \tl_set:Nn \l_tmpa_tl { { #2 } }
9538     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9539     { \@@_for_a_block:nnnnn ##1 }
9540     \@@_all_the_cells:
9541     \group_end:

```

```

9542     }
9543     { \@@_error:n { TikzEveryCell~without~tikz } }
9544 }
9545
9546
9547 \cs_new_protected:Nn \@@_all_the_cells:
9548 {
9549     \int_step_inline:nn \c@iRow
9550     {
9551         \int_step_inline:nn \c@jCol
9552         {
9553             \cs_if_exist:cF { cell - ##1 - #####1 }
9554             {
9555                 \clist_if_in:NcF \l_@@_corners_cells_clist
9556                 { ##1 - #####1 }
9557                 {
9558                     \bool_set_false:N \l_tmpa_bool
9559                     \cs_if_exist:cTF
9560                     { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
9561                     {
9562                         \bool_if:NF \l_@@_empty_bool
9563                         { \bool_set_true:N \l_tmpa_bool }
9564                     }
9565                     {
9566                         \bool_if:NF \l_@@_not_empty_bool
9567                         { \bool_set_true:N \l_tmpa_bool }
9568                     }
9569                     \bool_if:NT \l_tmpa_bool
9570                     {
9571                         \@@_block_tikz:nnnnn
9572                         \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
9573                     }
9574                 }
9575             }
9576         }
9577     }
9578 }
9579
9580 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9581 {
9582     \bool_if:NF \l_@@_empty_bool
9583     {
9584         \@@_block_tikz:nnnnn
9585         \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9586     }
9587     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9588 }
9589
9590 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9591 {
9592     \int_step_inline:nnn { #1 } { #3 }
9593     {
9594         \int_step_inline:nnn { #2 } { #4 }
9595         { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9596     }
9597 }

```

37 The command \ShowCellNames

```

9598 \NewDocumentCommand \@@_ShowCellNames { }
9599 {
9600     \bool_if:NT \l_@@_in_code_after_bool

```

```

9601 {
9602   \pgfpicture
9603   \pgfrememberpicturepositiononpagetrue
9604   \pgf@relevantforpicturesizefalse
9605   \pgfpathrectanglecorners
9606     { \@@_qpoint:n { 1 } }
9607     {
9608       \@@_qpoint:n
9609       { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
9610     }
9611   \pgfsetfillopacity { 0.75 }
9612   \pgfsetfillcolor { white }
9613   \pgfusepathqfill
9614   \endpgfpicture
9615 }
9616 \dim_gzero_new:N \g_@@_tmpc_dim
9617 \dim_gzero_new:N \g_@@_tmpd_dim
9618 \dim_gzero_new:N \g_@@_tmpe_dim
9619 \int_step_inline:nn { \c@iRow }
9620 {
9621   \bool_if:NTF \l_@@_in_code_after_bool
9622   {
9623     \pgfpicture
9624     \pgfrememberpicturepositiononpagetrue
9625     \pgf@relevantforpicturesizefalse
9626   }
9627   { \begin { pgfpicture } }
9628   \@@_qpoint:n { row - ##1 }
9629   \dim_set_eq:NN \l_tmpa_dim \pgf@y
9630   \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9631   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9632   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9633   \bool_if:NTF \l_@@_in_code_after_bool
9634     { \endpgfpicture }
9635     { \end { pgfpicture } }
9636   \int_step_inline:nn { \c@jCol }
9637   {
9638     \hbox_set:Nn \l_tmpa_box
9639     {
9640       \normalfont \Large \sffamily \bfseries
9641       \bool_if:NTF \l_@@_in_code_after_bool
9642         { \color { red } }
9643         { \color { red ! 50 } }
9644       ##1 - ####1
9645     }
9646     \bool_if:NTF \l_@@_in_code_after_bool
9647     {
9648       \pgfpicture
9649       \pgfrememberpicturepositiononpagetrue
9650       \pgf@relevantforpicturesizefalse
9651     }
9652     { \begin { pgfpicture } }
9653     \@@_qpoint:n { col - ####1 }
9654     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9655     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9656     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9657     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9658     \bool_if:NTF \l_@@_in_code_after_bool
9659       { \endpgfpicture }
9660       { \end { pgfpicture } }
9661     \fp_set:Nn \l_tmpa_fp
9662     {
9663       \fp_min:nn

```

```

9664         {
9665             \fp_min:nn
9666             { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9667             { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9668         }
9669         { 1.0 }
9670     }
9671     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9672     \pgfpicture
9673     \pgfrememberpicturepositiononpagetrue
9674     \pgf@relevantforpicturesizefalse
9675     \pgftransformshift
9676     {
9677         \pgfpoint
9678         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9679         { \dim_use:N \g_tmpa_dim }
9680     }
9681     \pgfnode
9682     { rectangle }
9683     { center }
9684     { \box_use:N \l_tmpa_box }
9685     { }
9686     { }
9687     \endpgfpicture
9688 }
9689 }
9690 }

```

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.

```

9691 \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.

```

9692 \bool_new:N \g_@@_footnote_bool
9693 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9694 {
9695     You-have-used-the~key~' \l_keys_key_str '~when~loading~nicematrix~
9696     but~that~key~is~unknown. \\
9697     It~will~be~ignored. \\
9698     For~a~list~of~the~available~keys,~type~H~<return>.
9699 }
9700 {
9701     The~available~keys~are~(in~alphabetic~order):~
9702     footnote,~
9703     footnotehyper,~
9704     messages-for-Overleaf,~
9705     renew-dots~and~
9706     renew-matrix.
9707 }
9708 \keys_define:nn { nicematrix }
9709 {
9710     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,

```

```

9711     renew-dots .value_forbidden:n = true ,
9712     renew-matrix .code:n = \@@_renew_matrix: ,
9713     renew-matrix .value_forbidden:n = true ,
9714     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9715     footnote .bool_set:N = \g_@@_footnote_bool ,
9716     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9717     unknown .code:n = \@@_error:n { Unknown-key-for-package }
9718   }
9719   \ProcessKeyOptions

9720   \@@_msg_new:nn { footnote-with-footnotehyper-package }
9721   {
9722     You-can't-use-the-option-'footnote'~because-the-package~
9723     footnotehyper-has-already-been-loaded.~
9724     If-you-want,-you-can-use-the-option-'footnotehyper'~and-the-footnotes~
9725     within-the-environments-of-nicematrix-will-be-extracted-with-the-tools~
9726     of-the-package-footnotehyper.\\
9727     The-package-footnote-won't-be-loaded.
9728   }

9729   \@@_msg_new:nn { footnotehyper-with-footnote-package }
9730   {
9731     You-can't-use-the-option-'footnotehyper'~because-the-package~
9732     footnote-has-already-been-loaded.~
9733     If-you-want,-you-can-use-the-option-'footnote'~and-the-footnotes~
9734     within-the-environments-of-nicematrix-will-be-extracted-with-the-tools~
9735     of-the-package-footnote.\\
9736     The-package-footnotehyper-won't-be-loaded.
9737   }

9738   \bool_if:NT \g_@@_footnote_bool
9739   {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9740     \IfClassLoadedTF { beamer }
9741     { \bool_set_false:N \g_@@_footnote_bool }
9742     {
9743       \IfPackageLoadedTF { footnotehyper }
9744       { \@@_error:n { footnote-with-footnotehyper-package } }
9745       { \usepackage { footnote } }
9746     }
9747   }

9748   \bool_if:NT \g_@@_footnotehyper_bool
9749   {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9750     \IfClassLoadedTF { beamer }
9751     { \bool_set_false:N \g_@@_footnote_bool }
9752     {
9753       \IfPackageLoadedTF { footnote }
9754       { \@@_error:n { footnotehyper-with-footnote-package } }
9755       { \usepackage { footnotehyper } }
9756     }
9757     \bool_set_true:N \g_@@_footnote_bool
9758   }

```

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.

```
9759 \bool_new:N \l_@@_underscore_loaded_bool
9760 \IfPackageLoadedT { underscore }
9761 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9762 \hook_gput_code:nnn { begindocument } { . }
9763 {
9764   \bool_if:NF \l_@@_underscore_loaded_bool
9765   {
9766     \IfPackageLoadedT { underscore }
9767     { \@@_error:n { underscore-after-nicematrix } }
9768   }
9769 }
```

40 Error messages of the package

When there is a unknown key, we try a “normal form” of the key and, when that normal form exists, we add that information in the error message.

The normal form is the lower case form of the key, with all the spaces replaced by hyphens (there is never spaces in the keys of nicematrix).

#1 is a clist of names of sets of keys and #2 is the error message to send.

```
9770 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
9771 {
9772   \str_set_eq:NN \l_tmpa_str \l_keys_key_str
9773   \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
9774   \str_set:Ne \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
9775   \bool_set_false:N \l_tmpa_bool
9776   \clist_map_inline:nn { #1 }
9777   {
9778     \keys_if_exist:neT { ##1 } { \l_tmpa_str }
9779     {
9780       \@@_error:n { key-with-normal-form-exists }
9781       \bool_set_true:N \l_tmpa_bool
9782       \clist_map_break:
9783     }
9784   }
9785   \bool_if:NF \l_tmpa_bool { \@@_error:n { #2 } }
9786 }
9787 \@@_msg_new:nn { key-with-normal-form-exists }
9788 {
9789   The~key~'\l_keys_key_str'~does~not~exists.~It~will~be~ignored.\\
9790   Maybe~you~want~to~use~the~key~'\l_tmpa_str'.
9791 }
9792 \str_const:Ne \c_@@_available_keys_str
9793 {
9794   \bool_if:nT { ! \g_@@_messages_for_Overleaf_bool }
9795   { For~a~list~of~the~available~keys,~type~H~<return>. }
9796 }
9797 \seq_new:N \g_@@_types_of_matrix_seq
9798 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9799 {
9800   NiceMatrix ,
```

```

9801     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9802 }
9803 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9804 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_err_too_many_cols:` is triggered. This command raises an error but also tries to give the best information to the user in the error message. The command `\seq_if_in:NoF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

9805 \cs_new_protected:Npn \@@_err_too_many_cols:
9806 {
9807   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9808   { \@@_fatal:nn { too-many-cols-for-array } }
9809   \int_compare:nNnT { \l_@@_last_col_int } = { -2 }
9810   { \@@_fatal:n { too-many-cols-for-matrix } }
9811   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
9812   { \@@_fatal:n { too-many-cols-for-matrix } }
9813   \bool_if:NF \l_@@_last_col_without_value_bool
9814   { \@@_fatal:n { too-many-cols-for-matrix-with-last-col } }
9815 }

```

The following command must *not* be protected since it's used in an error message.

```

9816 \cs_new:Npn \@@_message_hdotsfor:
9817 {
9818   \tl_if_empty:oF \g_@@_HVDotsfor_lines_tl
9819   { ~Maybe-your-use-of~ \token_to_str:N \Hdotsfor \ or~
9820     \token_to_str:N \Hbrace \ is-incorrect. }
9821 }

```

```

9822 \cs_new_protected:Npn \@@_Hline_in_cell:
9823 { \@@_fatal:n { Misuse-of-Hline } }
9824 \@@_msg_new:nn { Misuse-of-Hline }
9825 {
9826   Misuse-of-Hline. \\
9827   \token_to_str:N \Hline\ must-be-used-only-at-the-beginning-of-a-row.\\
9828   That-error-is-fatal.
9829 }
9830 \@@_msg_new:nn { hvlines,~rounded-corners-and-corners }
9831 {
9832   Incompatible-options.\\
9833   You-should-not-use-'hvlines',~'rounded-corners'~and-'corners'~at-the-same-time.\\
9834   The-output-will-not-be-reliable.
9835 }
9836 \@@_msg_new:nn { Body-alone }
9837 {
9838   \token_to_str:N \Body\ alone. \\
9839   You-have-used-\token_to_str:N \Body\ without-\token_to_str:N \CodeBefore.\\
9840   That-error-is-fatal.
9841 }
9842 \@@_msg_new:nn { key-color-inside }
9843 {
9844   Deleted-key.\\
9845   The-key-'color-inside'~(and-its-alias-'colortbl-like')~has-been-deleted-in
9846   ~'nicematrix'~and-must-not-be-used.\\
9847   This-error-is-fatal.
9848 }
9849 \@@_msg_new:nn { invalid-weight }
9850 {
9851   Unknown-key.\\
9852   The-key~' \l_keys_key_str '~of-your-column-X-is-unknown-and-will-be-ignored.
9853 }

```



```

9854 \@@_msg_new:nn { last~col~not~used }
9855 {
9856   Column~not~used.\\
9857   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
9858   in~your~\@@_full_name_env: .~
9859   However,~you~can~go~on.
9860 }
9861 \@@_msg_new:nn { too~many~cols~for~matrix~with~last~col }
9862 {
9863   Too~many~columns.\\
9864   In~the~row~ \int_eval:n { \c@iRow },~
9865   you~try~to~use~more~columns~
9866   than~allowed~by~your~ \@@_full_name_env: .
9867   \@@_message_hdotsfor: \
9868   The~maximal~number~of~columns~is~ \int_eval:n { \l_@@_last_col_int - 1 }~
9869   (plus~the~exterior~columns).~This~error~is~fatal.
9870 }
9871 \@@_msg_new:nn { too~many~cols~for~matrix }
9872 {
9873   Too~many~columns.\\
9874   In~the~row~ \int_eval:n { \c@iRow },~
9875   you~try~to~use~more~columns~than~allowed~by~your~ \@@_full_name_env: .
9876   \@@_message_hdotsfor: \
9877   Recall~that~the~maximal~number~of~columns~for~a~matrix~
9878   (excepted~the~potential~exterior~columns)~is~fixed~by~the~
9879   LaTeX~counter~'MaxMatrixCols'.~
9880   Its~current~value~is~ \int_use:N \c@MaxMatrixCols \
9881   (use~ \token_to_str:N \setcounter \ to~change~that~value).~
9882   This~error~is~fatal.
9883 }
9884 \@@_msg_new:nn { too~many~cols~for~array }
9885 {
9886   Too~many~columns.\\
9887   In~the~row~ \int_eval:n { \c@iRow },~
9888   ~you~try~to~use~more~columns~than~allowed~by~your~
9889   \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal~number~of~columns~is~
9890   \int_use:N \g_@@_static_num_of_col_int \
9891   \bool_if:nT
9892     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
9893     { ~(plus~the~exterior~ones) }
9894   since~the~preamble~is~' \g_@@_user_preamble_tl '.\\
9895   This~error~is~fatal.
9896 }
9897 \@@_msg_new:nn { columns~not~used }
9898 {
9899   Columns~not~used.\\
9900   The~preamble~of~your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '.~
9901   It~announces~ \int_use:N \g_@@_static_num_of_col_int \
9902   columns~but~you~only~used~ \int_use:N \c@jCol .\\
9903   The~columns~you~did~not~used~won't~be~created.\\
9904   You~won't~have~similar~warning~till~the~end~of~the~document.
9905 }
9906 \@@_msg_new:nn { empty~preamble }
9907 {
9908   Empty~preamble.\\
9909   The~preamble~of~your~ \@@_full_name_env: \ is~empty.\\
9910   This~error~is~fatal.
9911 }
9912 \@@_msg_new:nn { in~first~col }
9913 {

```

```

9914     Erroneous~use.\\
9915     You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
9916     That~command~will~be~ignored.
9917 }
9918 \\@@_msg_new:nn { in~last~col }
9919 {
9920     Erroneous~use.\\
9921     You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9922     That~command~will~be~ignored.
9923 }
9924 \\@@_msg_new:nn { in~first~row }
9925 {
9926     Erroneous~use.\\
9927     You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9928     That~command~will~be~ignored.
9929 }
9930 \\@@_msg_new:nn { in~last~row }
9931 {
9932     Erroneous~use.\\
9933     You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9934     That~command~will~be~ignored.
9935 }
9936 \\@@_msg_new:nn { TopRule~without~booktabs }
9937 {
9938     Erroneous~use.\\
9939     You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
9940     That~command~will~be~ignored.
9941 }
9942 \\@@_msg_new:nn { TopRule~without~tikz }
9943 {
9944     Erroneous~use.\\
9945     You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
9946     That~command~will~be~ignored.
9947 }
9948 \\@@_msg_new:nn { caption~outside~float }
9949 {
9950     Key~caption~forbidden.\\
9951     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9952     environment~(such~as~\\{table\\}).~This~key~will~be~ignored.
9953 }
9954 \\@@_msg_new:nn { short~caption~without~caption }
9955 {
9956     You~should~not~use~the~key~'short~caption'~without~'caption'.~
9957     However,~your~'short~caption'~will~be~used~as~'caption'.
9958 }
9959 \\@@_msg_new:nn { double~closing~delimiter }
9960 {
9961     Double~delimiter.\\
9962     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9963     delimiter.~This~delimiter~will~be~ignored.
9964 }
9965 \\@@_msg_new:nn { delimiter~after~opening }
9966 {
9967     Double~delimiter.\\
9968     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9969     delimiter.~That~delimiter~will~be~ignored.
9970 }
9971 \\@@_msg_new:nn { bad~option~for~line~style }
9972 {

```

```

9973     Bad-line-style.\
9974     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
9975     is~'standard'.~That~key~will~be~ignored.
9976 }

9977 \@@_msg_new:nn { corners~with~no~cell~nodes }
9978 {
9979     Incompatible~keys.\
9980     You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
9981     is~in~force.\
9982     If~you~go~on,~that~key~will~be~ignored.
9983 }

9984 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
9985 {
9986     Incompatible~keys.\
9987     You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
9988     is~in~force.\
9989     If~you~go~on,~those~extra~nodes~won't~be~created.
9990 }

9991 \@@_msg_new:nn { Identical~notes~in~caption }
9992 {
9993     Identical~tabular~notes.\
9994     You~can't~put~several~notes~with~the~same~content~in~
9995     \token_to_str:N \caption \ (but~you~can~in~the~main~tabular).\
9996     If~you~go~on,~the~output~will~probably~be~erroneous.
9997 }

9998 \@@_msg_new:nn { tabularnote~below~the~tabular }
9999 {
10000     \token_to_str:N \tabularnote \ forbidden\
10001     You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
10002     of~your~tabular~because~the~caption~will~be~composed~below~
10003     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
10004     key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .\
10005     Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
10006     no~similar~error~will~raised~in~this~document.
10007 }

10008 \@@_msg_new:nn { Unknown~key~for~rules }
10009 {
10010     Unknown~key.\
10011     There~is~only~two~keys~available~here:~width~and~color.\
10012     Your~key~' \l_keys_key_str ' ~will~be~ignored.
10013 }

10014 \@@_msg_new:nn { Unknown~key~for~Hbrace }
10015 {
10016     Unknown~key.\
10017     You~have~used~the~key~' \l_keys_key_str ' ~but~the~only~
10018     keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
10019     and~ \token_to_str:N \Vbrace \ are:~'color',~
10020     'horizontal~label(s)',~'shorten'~'shorten~end'~
10021     and~'shorten~start'~.\
10022     That~error~is~fatal.
10023 }

10024 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
10025 {
10026     Unknown~key.\
10027     There~is~only~two~keys~available~here:~
10028     'empty'~and~'not~empty'~.\
10029     Your~key~' \l_keys_key_str ' ~will~be~ignored.
10030 }

10031 \@@_msg_new:nn { Unknown~key~for~rotate }
10032 {

```

```

10033     Unknown~key.\\
10034     The~key~available~here~is~'c'.\\
10035     Your~key~' \l_keys_key_str '~will~be~ignored.
10036 }
10037 \@@_msg_new:nnn { Unknown~key~for~custom~line }
10038 {
10039     Unknown~key.\\
10040     The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
10041     It~you~go~on,~you~will~probably~have~other~errors. \\
10042     \c_@@_available_keys_str
10043 }
10044 {
10045     The~available~keys~are~(in~alphabetic~order):~
10046     ccommand,~
10047     color,~
10048     command,~
10049     dotted,~
10050     letter,~
10051     multiplicity,~
10052     sep-color,~
10053     tikz,~and~total~width.
10054 }
10055 \@@_msg_new:nnn { Unknown~key~for~xdots }
10056 {
10057     Unknown~key.\\
10058     The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
10059     \c_@@_available_keys_str
10060 }
10061 {
10062     The~available~keys~are~(in~alphabetic~order):~
10063     'color',~
10064     'horizontal(s)-labels',~
10065     'inter',~
10066     'line-style',~
10067     'radius',~
10068     'shorten',~
10069     'shorten-end'~and~'shorten-start'.
10070 }
10071 \@@_msg_new:nn { Unknown~key~for~rowcolors }
10072 {
10073     Unknown~key.\\
10074     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
10075     (and~you~try~to~use~' \l_keys_key_str '~)\\
10076     That~key~will~be~ignored.
10077 }
10078 \@@_msg_new:nn { label~without~caption }
10079 {
10080     You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
10081     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
10082 }
10083 \@@_msg_new:nn { W~warning }
10084 {
10085     Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
10086     (row~ \int_use:N \c@iRow ).
10087 }
10088 \@@_msg_new:nn { Construct~too~large }
10089 {
10090     Construct~too~large.\\
10091     Your~command~ \token_to_str:N #1
10092     can't~be~drawn~because~your~matrix~is~too~small.\\
10093     That~command~will~be~ignored.
10094 }

```

```

10095 \@@_msg_new:nn { underscore~after~nicematrix }
10096 {
10097   Problem~with~'underscore'.\\
10098   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
10099   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
10100   ' \token_to_str:N \Cdots \token_to_str:N _
10101   \{ n \token_to_str:N \text \{ ~times \} \}' .
10102 }
10103 \@@_msg_new:nn { ampersand~in~light-syntax }
10104 {
10105   Ampersand~forbidden.\\
10106   You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
10107   the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
10108 }
10109 \@@_msg_new:nn { double~backslash~in~light-syntax }
10110 {
10111   Double~backslash~forbidden.\\
10112   You~can't~use~ \token_to_str:N \\
10113   ~to~separate~rows~because~the~key~'light-syntax'~
10114   is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
10115   (set~by~the~key~'end-of-row').~This~error~is~fatal.
10116 }
10117 \@@_msg_new:nn { hlines~with~color }
10118 {
10119   Incompatible~keys.\\
10120   You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
10121   \token_to_str:N \Block \ when~the~key~'color'~or~'draw'~is~used.\\
10122   However,~you~can~put~several~commands~ \token_to_str:N \Block.\\
10123   Your~key~will~be~discarded.
10124 }
10125 \@@_msg_new:nn { bad~value~for~baseline }
10126 {
10127   Bad~value~for~baseline.\\
10128   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
10129   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
10130   \int_use:N \g_@@_row_total_int \ or~equal~to~'t',~'c'~or~'b'~or~of~
10131   the~form~'line-i'.\\
10132   A~value~of~1~will~be~used.
10133 }
10134 \@@_msg_new:nn { bad~value~for~baseline~line }
10135 {
10136   Bad~value~for~baseline~with~line.\\
10137   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
10138   valid.~The~number~of~the~line~must~be~between~1~and~
10139   \int_eval:n { \c@iRow + 1 } \\
10140   A~value~of~'line-1'~will~be~used.
10141 }
10142 \@@_msg_new:nn { detection~of~empty~cells }
10143 {
10144   Problem~with~'not~empty'\\
10145   For~technical~reasons,~you~must~activate~
10146   'create~cell~nodes'~in~ \token_to_str:N \CodeBefore \
10147   in~order~to~use~the~key~' \l_keys_key_str '.\\
10148   That~key~will~be~ignored.
10149 }
10150 \@@_msg_new:nn { siunitx~not~loaded }
10151 {
10152   siunitx~not~loaded\\
10153   You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
10154   That~error~is~fatal.
10155 }

```

```

10156 \@@_msg_new:nn { Invalid-name }
10157 {
10158   Invalid-name.\
10159   You-can't-give-the-name~' \l_keys_value_tl '~to-a~ \token_to_str:N
10160   \SubMatrix \ of~your~ \@@_full_name_env: .\
10161   A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\
10162   This-key-will-be-ignored.
10163 }

10164 \@@_msg_new:nn { Hbrace-not-allowed }
10165 {
10166   Command-not-allowed.\
10167   You-can't-use-the-command~ \token_to_str:N #1
10168   because-you-have-not-loaded~
10169   \IfPackageLoadedTF { tikz }
10170     { the-TikZ-library~'decorations.pathreplacing'~Use~ }
10171     { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
10172   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \
10173   That-command-will-be-ignored.
10174 }

10175 \@@_msg_new:nn { Vbrace-not-allowed }
10176 {
10177   Command-not-allowed.\
10178   You-can't-use-the-command~ \token_to_str:N \Vbrace \
10179   because-you-have-not-loaded-TikZ~
10180   and-the-TikZ-library~'decorations.pathreplacing'.\
10181   Use: ~\token_to_str:N \usepackage \{tikz\}~
10182   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \
10183   That-command-will-be-ignored.
10184 }

10185 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
10186 {
10187   Wrong-line.\
10188   You-try-to-draw-a-#1~line-of-number~'#2'~in-a~
10189   \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but-that~
10190   number-is-not-valid.~It-will-be-ignored.
10191 }

10192 \@@_msg_new:nn { Impossible-delimiter }
10193 {
10194   Impossible-delimiter.\
10195   It's-impossible-to-draw-the-#1~delimiter-of~your~
10196   \token_to_str:N \SubMatrix \ because-all-the-cells-are-empty~
10197   in-that-column.
10198   \bool_if:NT \l_@@_submatrix_slim_bool
10199     { ~Maybe-you-should-try-without-the-key~'slim'. } \
10200   This~ \token_to_str:N \SubMatrix \ will-be-ignored.
10201 }

10202 \@@_msg_new:nnn { width-without-X-columns }
10203 {
10204   You-have-used-the-key~'width'~but-you-have-put-no~'X'~column-in~
10205   the-preamble~(' \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .\
10206   That-key-will-be-ignored.
10207 }
10208 {
10209   This-message-is-the-message~'width-without-X-columns'~
10210   of-the-module~'nicematrix'.~
10211   The-experimented-users-can-disable-that-message-with~
10212   \token_to_str:N \msg_redirect_name:nnn .\
10213 }

10214

10215 \@@_msg_new:nn { key-multiplicity-with-dotted }
10216 {

```

```

10217 Incompatible-keys. \\
10218 You-have-used-the-key~'multiplicity'~with~the-key~'dotted'~
10219 in~a~'custom-line'~.~They-are-incompatible. \\
10220 The-key~'multiplicity'~will-be-discarded.
10221 }
10222 \@@_msg_new:nn { empty-environment }
10223 {
10224   Empty-environment.\\
10225   Your~ \@@_full_name_env: \ is-empty.~This-error-is-fatal.
10226 }
10227 \@@_msg_new:nn { No-letter-and-no-command }
10228 {
10229   Erroneous-use.\\
10230   Your-use-of~'custom-line'~is-no-op~since-you-don't-have-used-the~
10231 key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10232 ~'ccommand'~(to~draw~horizontal~rules).\\
10233   However,~you~can~go~on.
10234 }
10235 \@@_msg_new:nn { Forbidden-letter }
10236 {
10237   Forbidden-letter.\\
10238   You-can't-use-the-letter~'#1'~for~a~customized-line.~
10239   It-will-be-ignored.\\
10240   The-forbidden-letters-are:~\c_@@_forbidden_letters_str
10241 }
10242 \@@_msg_new:nn { Several-letters }
10243 {
10244   Wrong-name.\\
10245   You-must-use-only-one-letter~as~value~for~the~key~'letter'~(and-you~
10246 have-used~' \l_@@_letter_str ').\\
10247   It-will-be-ignored.
10248 }
10249 \@@_msg_new:nn { Delimiter-with-small }
10250 {
10251   Delimiter-forbidden.\\
10252   You-can't-put-a-delimiter~in~the-preamble-of~your~
10253 \@@_full_name_env: \
10254 because-the-key~'small'~is-in-force.\\
10255   This-error-is-fatal.
10256 }
10257 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
10258 {
10259   Unknown-cell.\\
10260   Your-command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10261 the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10262 can't-be-executed~because~a~cell~doesn't~exist.\\
10263   This-command~ \token_to_str:N \line \ will-be-ignored.
10264 }
10265 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
10266 {
10267   Duplicate-name.\\
10268   The-name~'#1'~is-already-used~for~a~ \token_to_str:N \SubMatrix \
10269 in~this~ \@@_full_name_env: .\\
10270   This-key-will-be-ignored.\\
10271   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10272     { For~a~list~of~the~names~already~used,~type-H~<return>. }
10273 }
10274 {
10275   The-names~already~defined~in~this~ \@@_full_name_env: \ are:~
10276 \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10277 }

```

```

10278 \@@_msg_new:nn { r~or~l~with~preamble }
10279 {
10280   Erroneous~use.\\
10281   You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
10282   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10283   your~ \@@_full_name_env: .\\
10284   This~key~will~be~ignored.
10285 }
10286 \@@_msg_new:nn { Hdotsfor~in~col~0 }
10287 {
10288   Erroneous~use.\\
10289   You~can't~use~ \token_to_str:N \Hdotsfor \ in~an~exterior~column~of~
10290   the~array.~This~error~is~fatal.
10291 }
10292 \@@_msg_new:nn { bad~corner }
10293 {
10294   Bad~corner.\\
10295   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10296   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
10297   This~specification~of~corner~will~be~ignored.
10298 }
10299 \@@_msg_new:nn { bad~border }
10300 {
10301   Bad~border.\\
10302   \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
10303   (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
10304   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10305   also~use~the~key~'tikz'
10306   \IfPackageLoadedF { tikz }
10307   { ~if~you~load~the~LaTeX~package~'tikz' } ).\\
10308   This~specification~of~border~will~be~ignored.
10309 }
10310 \@@_msg_new:nn { TikzEveryCell~without~tikz }
10311 {
10312   TikZ~not~loaded.\\
10313   You~can't~use~ \token_to_str:N \TikzEveryCell \
10314   because~you~have~not~loaded~tikz.~
10315   This~command~will~be~ignored.
10316 }
10317 \@@_msg_new:nn { tikz~key~without~tikz }
10318 {
10319   TikZ~not~loaded.\\
10320   You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N
10321   \Block '~because~you~have~not~loaded~tikz.~
10322   This~key~will~be~ignored.
10323 }
10324 \@@_msg_new:nn { Bad~argument~for~Block }
10325 {
10326   Bad~argument.\\
10327   The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
10328   be~of~the~form~'i-j'~(or~completely~empty)~and~you~have~used:~
10329   '#1'. \\
10330   If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono-cell~(as~if~
10331   the~argument~was~empty).
10332 }
10333 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
10334 {
10335   Erroneous~use.\\
10336   In~the~ \@@_full_name_env: ,~you~must~use~the~key~
10337   'last~col'~without~value.\\
10338   However,~you~can~go~on~for~this~time~

```



```

10339     (the-value~' \l_keys_value_tl '~will-be-ignored).
10340 }

10341 \@@_msg_new:nn { last-col-non-empty-for~NiceMatrixOptions }
10342 {
10343     Erroneous~use. \\
10344     In~\token_to_str:N \NiceMatrixOptions ,~you-must-use-the-key~
10345     'last-col'~without-value. \\
10346     However,~you-can-go-on-for~this-time~
10347     (the-value~' \l_keys_value_tl '~will-be-ignored).
10348 }

10349 \@@_msg_new:nn { Block-too-large-1 }
10350 {
10351     Block~too-large. \\
10352     You-try-to-draw-a-block-in-the-cell~#1-#2-of~your~matrix-but-the-matrix-is~
10353     too-small~for~that-block. \\
10354     This-block-and-maybe-others-will-be-ignored.
10355 }

10356 \@@_msg_new:nn { Block-too-large-2 }
10357 {
10358     Block~too-large. \\
10359     The~preamble-of~your~ \@@_full_name_env: \ announces~ \int_use:N
10360     \g_@@_static_num_of_col_int \
10361     columns~but~you-use-only~ \int_use:N \c@jCol \ and~that's~why~a~block~
10362     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
10363     (&)~at~the~end~of~the~first~row~of~your~ \@@_full_name_env: . \\
10364     This~block~and~maybe~others~will~be~ignored.
10365 }

10366 \@@_msg_new:nn { unknown~column-type }
10367 {
10368     Bad~column~type. \\
10369     The~column~type~'#1'~in~your~ \@@_full_name_env: \
10370     is~unknown. \\
10371     This~error~is~fatal.
10372 }

10373 \@@_msg_new:nn { unknown~column-type-multicolumn }
10374 {
10375     Bad~column~type. \\
10376     The~column~type~'#1'~in~the~command~\token_to_str:N \multicolumn \
10377     ~of~your~ \@@_full_name_env: \
10378     is~unknown. \\
10379     This~error~is~fatal.
10380 }

10381 \@@_msg_new:nn { unknown~column-type-S }
10382 {
10383     Bad~column~type. \\
10384     The~column~type~'S'~in~your~ \@@_full_name_env: \ is~unknown. \\
10385     If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10386     load~that~package. \\
10387     This~error~is~fatal.
10388 }

10389 \@@_msg_new:nn { unknown~column-type-S-multicolumn }
10390 {
10391     Bad~column~type. \\
10392     The~column~type~'S'~in~the~command~\token_to_str:N \multicolumn \
10393     of~your~ \@@_full_name_env: \ is~unknown. \\
10394     If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10395     load~that~package. \\
10396     This~error~is~fatal.
10397 }

10398 \@@_msg_new:nn { tabularnote~forbidden }

```

```

10399 {
10400   Forbidden~command. \\
10401   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10402   ~here.~This~command~is~available~only~in~
10403   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
10404   the~argument~of~a~command~\token_to_str:N \caption \ included~
10405   in~an~environment~\{table\}. \\
10406   This~command~will~be~ignored.
10407 }
10408 \@@_msg_new:nn { borders~forbidden }
10409 {
10410   Forbidden~key.\\
10411   You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \
10412   because~the~option~'rounded-corners'~
10413   is~in~force~with~a~non-zero~value.\\
10414   This~key~will~be~ignored.
10415 }
10416 \@@_msg_new:nn { bottomrule~without~booktabs }
10417 {
10418   booktabs~not~loaded.\\
10419   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
10420   loaded~'booktabs'.\\
10421   This~key~will~be~ignored.
10422 }
10423 \@@_msg_new:nn { enumitem~not~loaded }
10424 {
10425   enumitem~not~loaded. \\
10426   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10427   ~because~you~haven't~loaded~'enumitem'. \\
10428   All~the~commands~ \token_to_str:N \tabularnote \ will~be~
10429   ignored~in~the~document.
10430 }
10431 \@@_msg_new:nn { tikz~without~tikz }
10432 {
10433   Tikz~not~loaded. \\
10434   You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
10435   loaded.~If~you~go~on,~that~key~will~be~ignored.
10436 }
10437 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
10438 {
10439   Tikz~not~loaded. \\
10440   You~have~used~the~key~'tikz'~in~the~definition~of~a~
10441   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
10442   You~can~go~on~but~you~will~have~another~error~if~you~actually~
10443   use~that~custom~line.
10444 }
10445 \@@_msg_new:nn { tikz~in~borders~without~tikz }
10446 {
10447   Tikz~not~loaded. \\
10448   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
10449   command~' \token_to_str:N \Block ')~but~tikz~is~not~loaded.~
10450   That~key~will~be~ignored.
10451 }
10452 \@@_msg_new:nn { color~in~custom~line~with~tikz }
10453 {
10454   Erroneous~use.\\
10455   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
10456   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
10457   The~key~'color'~will~be~discarded.
10458 }

```

```

10459 \@@_msg_new:nn { Wrong-last-row }
10460 {
10461   Wrong-number.\
10462   You-have-used-'last-row= \int_use:N \l_@@_last_row_int '~but-your~
10463   \@@_full_name_env: \ seems-to-have~ \int_use:N \c@iRow \ rows.~
10464   If-you-go-on,~the-value-of~ \int_use:N \c@iRow \ will-be-used-for~
10465   last-row-but-you-should-correct-your-code.~You-can-avoid-this~
10466   problem-by-using-'last-row'~without-value~(more-compilations~
10467   might-be-necessary).
10468 }

10469 \@@_msg_new:nn { Yet-in-env }
10470 {
10471   Nested-environments.\
10472   Environments-of~nicematrix~can't-be-nested.\
10473   This-error-is-fatal.
10474 }

10475 \@@_msg_new:nn { Outside-math-mode }
10476 {
10477   Outside-math-mode.\
10478   The-\@@_full_name_env: \ can-be-used-only-in-math-mode~
10479   (and-not-in~ \token_to_str:N \vcenter ).\
10480   This-error-is-fatal.
10481 }

10482 \@@_msg_new:nn { One-letter-allowed }
10483 {
10484   Bad-name.\
10485   The-value-of-key~' \l_keys_key_str '~must-be-of-length~1~and~
10486   you-have-used~' \l_keys_value_tl '~.\
10487   It-will-be-ignored.
10488 }

10489 \@@_msg_new:nn { TabularNote-in-CodeAfter }
10490 {
10491   Environment~\{TabularNote\}~forbidden.\
10492   You-must-use~\{TabularNote\}~at-the-end-of-your~\{NiceTabular\}~
10493   but~*before*~the~ \token_to_str:N \CodeAfter . \
10494   This-environment~\{TabularNote\}~will-be-ignored.
10495 }

10496 \@@_msg_new:nn { varwidth-not-loaded }
10497 {
10498   varwidth-not-loaded.\
10499   You-can't-use-the-column-type~'V'~because~'varwidth'~is-not~
10500   loaded.\
10501   Your~column~will-behave-like~'p'.
10502 }

10503 \@@_msg_new:nn { varwidth-not-loaded-in-X }
10504 {
10505   varwidth-not-loaded.\
10506   You-can't-use-the-key~'V'~in-your~column~'X'~
10507   because~'varwidth'~is-not-loaded.\
10508   It-will-be-ignored. \
10509 }

10510 \@@_msg_new:nnn { Unknown-key-for-RulesBis }
10511 {
10512   Unknown-key.\
10513   Your-key~' \l_keys_key_str '~is-unknown-for-a-rule.\
10514   \c_@@_available_keys_str
10515 }
10516 {
10517   The-available-keys-are~(in-alphabetic-order):~
10518   color,~
10519   dotted,~

```

```

10520     multiplicity,~
10521     sep-color,~
10522     tikz,~and~total-width.
10523 }
10524
10525 \@@_msg_new:nnn { Unknown~key~for~Block }
10526 {
10527     Unknown~key. \\
10528     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10529     \token_to_str:N \Block . \\
10530     It~will~be~ignored. \\
10531     \c_@@_available_keys_str
10532 }
10533 {
10534     The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10535     b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
10536     opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
10537     and~vlines.
10538 }
10539 \@@_msg_new:nnn { Unknown~key~for~Brace }
10540 {
10541     Unknown~key. \\
10542     The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
10543     \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
10544     It~will~be~ignored. \\
10545     \c_@@_available_keys_str
10546 }
10547 {
10548     The~available~keys~are~(in~alphabetic~order):~color,~left-shorten,~
10549     right-shorten,~shorten~(which~fixes~both~left-shorten~and~
10550     right-shorten)~and~yshift.
10551 }
10552 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10553 {
10554     Unknown~key. \\
10555     The~key~' \l_keys_key_str '~is~unknown. \\
10556     It~will~be~ignored. \\
10557     \c_@@_available_keys_str
10558 }
10559 {
10560     The~available~keys~are~(in~alphabetic~order):~
10561     delimiters/color,~
10562     rules~(with~the~subkeys~'color'~and~'width'),~
10563     sub-matrix~(several~subkeys)~
10564     and~xdots~(several~subkeys).~
10565     The~latter~is~for~the~command~ \token_to_str:N \line .
10566 }
10567 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10568 {
10569     Unknown~key. \\
10570     The~key~' \l_keys_key_str '~is~unknown. \\
10571     It~will~be~ignored. \\
10572     \c_@@_available_keys_str
10573 }
10574 {
10575     The~available~keys~are~(in~alphabetic~order):~
10576     create-cell-nodes,~
10577     delimiters/color~and~
10578     sub-matrix~(several~subkeys).
10579 }
10580 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10581 {

```

```

10582     Unknown~key.\\
10583     The~key~' \l_keys_key_str '~is~unknown.\\
10584     That~key~will~be~ignored. \\
10585     \c_@@_available_keys_str
10586 }
10587 {
10588     The~available~keys~are~(in~alphabetic~order):~
10589     'delimiters/color',~
10590     'extra-height',~
10591     'hlines',~
10592     'hvlines',~
10593     'left-xshift',~
10594     'name',~
10595     'right-xshift',~
10596     'rules'~(with~the~subkeys~'color'~and~'width'),~
10597     'slim',~
10598     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10599     and~'right-xshift').\\
10600 }
10601 \@@_msg_new:nnn { Unknown~key~for~notes }
10602 {
10603     Unknown~key.\\
10604     The~key~' \l_keys_key_str '~is~unknown.\\
10605     That~key~will~be~ignored. \\
10606     \c_@@_available_keys_str
10607 }
10608 {
10609     The~available~keys~are~(in~alphabetic~order):~
10610     bottomrule,~
10611     code-after,~
10612     code-before,~
10613     detect-duplicates,~
10614     enumitem-keys,~
10615     enumitem-keys-para,~
10616     para,~
10617     label-in-list,~
10618     label-in-tabular~and~
10619     style.
10620 }
10621 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10622 {
10623     Unknown~key.\\
10624     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10625     \token_to_str:N \RowStyle . \\
10626     That~key~will~be~ignored. \\
10627     \c_@@_available_keys_str
10628 }
10629 {
10630     The~available~keys~are~(in~alphabetic~order):~
10631     bold,~
10632     cell-space-top-limit,~
10633     cell-space-bottom-limit,~
10634     cell-space-limits,~
10635     color,~
10636     fill~(alias:~rowcolor),~
10637     nb-rows,~
10638     opacity~and~
10639     rounded-corners.
10640 }
10641 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10642 {
10643     Unknown~key.\\
10644     The~key~' \l_keys_key_str '~is~unknown~for~the~command~

```

```

10645 \token_to_str:N \NiceMatrixOptions . \\
10646 That~key~will~be~ignored. \\
10647 \c_@@_available_keys_str
10648 }
10649 {
10650 The~available~keys~are~(in~alphabetic~order):~
10651 &-in-blocks,~
10652 allow-duplicate-names,~
10653 ampersand-in-blocks,~
10654 caption-above,~
10655 cell-space-bottom-limit,~
10656 cell-space-limits,~
10657 cell-space-top-limit,~
10658 code-for-first-col,~
10659 code-for-first-row,~
10660 code-for-last-col,~
10661 code-for-last-row,~
10662 corners,~
10663 custom-key,~
10664 create-extra-nodes,~
10665 create-medium-nodes,~
10666 create-large-nodes,~
10667 custom-line,~
10668 delimiters~(several~subkeys),~
10669 end-of-row,~
10670 first-col,~
10671 first-row,~
10672 hlines,~
10673 hvlines,~
10674 hvlines-except-borders,~
10675 last-col,~
10676 last-row,~
10677 left-margin,~
10678 light-syntax,~
10679 light-syntax-expanded,~
10680 matrix/columns-type,~
10681 no-cell-nodes,~
10682 notes~(several~subkeys),~
10683 nullify-dots,~
10684 pgf-node-code,~
10685 renew-dots,~
10686 renew-matrix,~
10687 respect-arraystretch,~
10688 rounded-corners,~
10689 right-margin,~
10690 rules~(with~the~subkeys~'color'~and~'width'),~
10691 small,~
10692 sub-matrix~(several~subkeys),~
10693 vlines,~
10694 xdots~(several~subkeys).
10695 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

10696 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10697 {
10698 Unknown~key.\\
10699 The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
10700 \{NiceArray\}. \\
10701 That~key~will~be~ignored. \\
10702 \c_@@_available_keys_str
10703 }
10704 {
10705 The~available~keys~are~(in~alphabetic~order):~

```

```

10706 &-in-blocks,~
10707 ampersand-in-blocks,~
10708 b,~
10709 baseline,~
10710 c,~
10711 cell-space-bottom-limit,~
10712 cell-space-limits,~
10713 cell-space-top-limit,~
10714 code-after,~
10715 code-for-first-col,~
10716 code-for-first-row,~
10717 code-for-last-col,~
10718 code-for-last-row,~
10719 columns-width,~
10720 corners,~
10721 create-extra-nodes,~
10722 create-medium-nodes,~
10723 create-large-nodes,~
10724 extra-left-margin,~
10725 extra-right-margin,~
10726 first-col,~
10727 first-row,~
10728 hlines,~
10729 hvlines,~
10730 hvlines-except-borders,~
10731 last-col,~
10732 last-row,~
10733 left-margin,~
10734 light-syntax,~
10735 light-syntax-expanded,~
10736 name,~
10737 no-cell-nodes,~
10738 nullify-dots,~
10739 pgf-node-code,~
10740 renew-dots,~
10741 respect-arraystretch,~
10742 right-margin,~
10743 rounded-corners,~
10744 rules~(with~the~subkeys~'color'~and~'width'),~
10745 small,~
10746 t,~
10747 vlines,~
10748 xdots/color,~
10749 xdots/shorten-start,~
10750 xdots/shorten-end,~
10751 xdots/shorten-and~
10752 xdots/line-style.
10753 }

```

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`).

```

10754 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10755 {
10756   Unknown~key.\\
10757   The~key~' \l_keys_key_str '~is~unknown~for~the~
10758   \@@_full_name_env: . \\
10759   That~key~will~be~ignored. \\
10760   \c_@@_available_keys_str
10761 }
10762 {
10763   The~available~keys~are~(in~alphabetic~order):~
10764   &-in-blocks,~
10765   ampersand-in-blocks,~

```

```

10766     b,~
10767     baseline,~
10768     c,~
10769     cell-space-bottom-limit,~
10770     cell-space-limits,~
10771     cell-space-top-limit,~
10772     code-after,~
10773     code-for-first-col,~
10774     code-for-first-row,~
10775     code-for-last-col,~
10776     code-for-last-row,~
10777     columns-type,~
10778     columns-width,~
10779     corners,~
10780     create-extra-nodes,~
10781     create-medium-nodes,~
10782     create-large-nodes,~
10783     extra-left-margin,~
10784     extra-right-margin,~
10785     first-col,~
10786     first-row,~
10787     hlines,~
10788     hvlines,~
10789     hvlines-except-borders,~
10790     l,~
10791     last-col,~
10792     last-row,~
10793     left-margin,~
10794     light-syntax,~
10795     light-syntax-expanded,~
10796     name,~
10797     no-cell-nodes,~
10798     nullify-dots,~
10799     pgf-node-code,~
10800     r,~
10801     renew-dots,~
10802     respect-arraystretch,~
10803     right-margin,~
10804     rounded-corners,~
10805     rules~(with~the~subkeys~'color'~and~'width'),~
10806     small,~
10807     t,~
10808     vlines,~
10809     xdots/color,~
10810     xdots/shorten-start,~
10811     xdots/shorten-end,~
10812     xdots/shorten-and~
10813     xdots/line-style.
10814 }

10815 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10816 {
10817     Unknown~key.\\
10818     The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
10819     \{NiceTabular\}. \\
10820     That~key~will~be~ignored. \\
10821     \c_@@_available_keys_str
10822 }
10823 {
10824     The~available~keys~are~(in~alphabetic~order):~
10825     &~in~blocks,~
10826     ampersand~in~blocks,~
10827     b,~
10828     baseline,~

```



```

10829 c,~
10830 caption,~
10831 cell-space-bottom-limit,~
10832 cell-space-limits,~
10833 cell-space-top-limit,~
10834 code-after,~
10835 code-for-first-col,~
10836 code-for-first-row,~
10837 code-for-last-col,~
10838 code-for-last-row,~
10839 columns-width,~
10840 corners,~
10841 custom-line,~
10842 create-extra-nodes,~
10843 create-medium-nodes,~
10844 create-large-nodes,~
10845 extra-left-margin,~
10846 extra-right-margin,~
10847 first-col,~
10848 first-row,~
10849 hlines,~
10850 hvlines,~
10851 hvlines-except-borders,~
10852 label,~
10853 last-col,~
10854 last-row,~
10855 left-margin,~
10856 light-syntax,~
10857 light-syntax-expanded,~
10858 name,~
10859 no-cell-nodes,~
10860 notes~(several~subkeys),~
10861 nullify-dots,~
10862 pgf-node-code,~
10863 renew-dots,~
10864 respect-arraystretch,~
10865 right-margin,~
10866 rounded-corners,~
10867 rules~(with~the~subkeys~'color'~and~'width'),~
10868 short-caption,~
10869 t,~
10870 tabularnote,~
10871 vlines,~
10872 xdots/color,~
10873 xdots/shorten-start,~
10874 xdots/shorten-end,~
10875 xdots/shorten-and~
10876 xdots/line-style.
10877 }

10878 \@@_msg_new:nnn { Duplicate-name }
10879 {
10880   Duplicate-name.\
10881   The-name~' \l_keys_value_tl 'is-already-used-and-you-shouldn't-use~
10882   the-same-environment-name-twice.~You-can-go-on,~but,~
10883   maybe,~you-will-have-incorrect-results-especially~
10884   if-you-use~'columns-width=auto'.~If-you-don't-want-to-see-this~
10885   message-again,~use-the-key~'allow-duplicate-names'~in~
10886   ' \token_to_str:N \NiceMatrixOptions '\
10887   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10888   { For-a-list-of-the-names-already-used,~type-H~<return>. }
10889 }
10890 {
10891   The-names-already-defined-in-this-document-are:-

```

```

10892 \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
10893 }
10894 \@@_msg_new:nn { caption-above-in-env }
10895 {
10896   The-key~'caption-above'~must~be~used~in~\token_to_str:N \NiceMatrixOptions.\\
10897   That~key~will~be~ignored.
10898 }
10899 \@@_msg_new:nn { show-cell-names }
10900 {
10901   There~is~no~key~'show-cell-names'~in~nicematrix.\\
10902   You~should~use~the~command~\token_to_str:N \ShowCellNames\
10903   in~the~\token_to_str:N \CodeBefore\ or~the~\token_to_str:N
10904   \CodeAfter. \\
10905   That~key~will~be~ignored.
10906 }
10907 \@@_msg_new:nn { Option~auto~for~columns~width }
10908 {
10909   Erroneous~use.\\
10910   You~can't~give~the~value~'auto'~to~the~key~'columns~width'~here.~
10911   That~key~will~be~ignored.
10912 }
10913 \@@_msg_new:nn { NiceTabularX~without~X }
10914 {
10915   NiceTabularX~without~X.\\
10916   You~should~not~use~\{NiceTabularX\}~without~X~columns.\\
10917   However,~you~can~go~on.
10918 }
10919 \@@_msg_new:nn { Preamble~forgotten }
10920 {
10921   Preamble~forgotten.\\
10922   You~have~probably~forgotten~the~preamble~of~your~
10923   \@@_full_name_env: . \\
10924   This~error~is~fatal.
10925 }
10926 \@@_msg_new:nn { Invalid~col~number }
10927 {
10928   Invalid~column~number.\\
10929   A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
10930   specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.
10931 }
10932 \@@_msg_new:nn { Invalid~row~number }
10933 {
10934   Invalid~row~number.\\
10935   A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
10936   specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.
10937 }
10938 \@@_define_com:NNN p ( )
10939 \@@_define_com:NNN b [ ]
10940 \@@_define_com:NNN v | |
10941 \@@_define_com:NNN V \| \|
10942 \@@_define_com:NNN B \{ \}

```

Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	3
4	Parameters	9
5	The command <code>\tabularnote</code>	20
6	Command for creation of rectangle nodes	25
7	The options	26
8	Important code used by <code>{NiceArrayWithDelims}</code>	37
9	The <code>\CodeBefore</code>	51
10	The environment <code>{NiceArrayWithDelims}</code>	55
11	Construction of the preamble of the array	60
12	The redefinition of <code>\multicolumn</code>	76
13	The environment <code>{NiceMatrix}</code> and its variants	94
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	95
15	After the construction of the array	96
16	We draw the dotted lines	103
17	The actual instructions for drawing the dotted lines with Tikz	118
18	User commands available in the new environments	123
19	The command <code>\line</code> accessible in code-after	129
20	The command <code>\RowStyle</code>	131
21	Colors of cells, rows and columns	134
22	The vertical and horizontal rules	146
23	The empty corners	163
24	The environment <code>{NiceMatrixBlock}</code>	165
25	The extra nodes	166
26	The blocks	171
27	How to draw the dotted lines transparently	197
28	Automatic arrays	198
29	The redefinition of the command <code>\dotfill</code>	199
30	The command <code>\diagbox</code>	200

31	The keyword <code>\CodeAfter</code>	201
32	The delimiters in the preamble	202
33	The command <code>\SubMatrix</code>	203
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	212
35	The commands <code>HBrace</code> et <code>VBrace</code>	215
36	The command <code>TikzEveryCell</code>	218
37	The command <code>\ShowCellNames</code>	219
38	We process the options at package loading	221
39	About the package underscore	223
40	Error messages of the package	223