

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

February 22, 2026

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

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

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

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

*This document corresponds to the version 7.7 of `nicematrix`, at the date of 2026/02/22.

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   {
35     \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 }

```

We keep also in memory in another message the complement of information (generally the list of the available keys) in order to write it the log file in all circumstances (it will be useful for the AI of some systems such as Prism).

```

36     \msg_new:nnn { nicematrix } { #1~+ } { #3 }
37   }
38 }

```

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

```

39 \cs_new_protected:Npn \@@_error_or_warning:n
40 {
41   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
42   \@@_warning:n
43   \@@_error:n
44 }

```

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”.

```

45 \bool_new:N \g_@@_messages_for_Overleaf_bool
46 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
47 {
48   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
49   || \str_if_eq_p:ee \c_sys_jobname_str { output } % for Overleaf
50 }

51 \@@_msg_new:nn { mdwtab-loaded }
52 {
53   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
54   This~error~is~fatal.
55 }

56 \hook_gput_code:nnn { begindocument / end } { . }
57 { \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`).

```

58 \cs_new_protected:Npn \@@_collect_options:n #1
59 {
60   \peek_meaning:NTF [
61     { \@@_collect_options:nw { #1 } }
62     { #1 { } }
63   }

```

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

```

64 \NewDocumentCommand \@@_collect_options:nw { m r[] }
65 { \@@_collect_options:nn { #1 } { #2 } }
66
67 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
68 {
69   \peek_meaning:NTF [
70     { \@@_collect_options:nnw { #1 } { #2 } }
71     { #1 { #2 } }
72   }
73
74 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
75 { \@@_collect_options:nn { #1 } { #2 , #3 } }

```

3 Technical definitions

Here are definitions that have been added to the LaTeX kernel in February 2006.

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

```

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

84 \tl_const:Nn \c_@@_brace_tl { nicematrix/brace }
85 \tl_const:Nn \c_@@_mirrored_brace_tl { nicematrix/mirrored-brace }

```

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

```

86 \tl_new:N \l_@@_argspec_tl

```

```

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

95 \hook_gput_code:nnn { begindocument } { . }
96 {
97   \IfPackageLoadedTF { tikz }
98   {

```

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.

```

99   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
100   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
101 }
102 {
103   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
104   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
105 }
106 }

```

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`).

```

107 \IfClassLoadedTF { revtex4-1 }
108 { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
109 {
110   \IfClassLoadedTF { revtex4-2 }
111   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
112   {

```

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

```

113   \cs_if_exist:NT \rvtx@ifformat@geq
114   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
115   { \bool_const:Nn \c_@@_revtex_bool { \c_false_bool } }
116 }
117 }

```

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.

```

118 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
119 {
120   \iow_now:Nn \@mainaux
121   {
122     \ExplSyntaxOn
123     \cs_if_free:NT \pgfsyspdfmark
124     { \cs_set_eq:NN \pgfsyspdfmark \gobblethree }
125     \ExplSyntaxOff
126   }
127   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
128 }

```

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

```

129 \ProvideDocumentCommand \iddots { }
130 {
131   \mathinner
132   {
133     \mkern 1 mu
134     \box_move_up:nn { 1 pt } { \hbox { . } }
135     \mkern 2 mu
136     \box_move_up:nn { 4 pt } { \hbox { . } }
137     \mkern 2 mu
138     \box_move_up:nn { 7 pt }
139     { \vbox:n { \kern 7 pt \hbox { . } } }
140     \mkern 1 mu
141   }
142 }

```

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.

```

143 \hook_gput_code:nnn { begindocument } { . }
144 {
145   \IfPackageLoadedT { booktabs }
146   {
147     \iow_now:Nn \@mainaux
148     {
149       \ExplSyntaxOn
150       \cs_if_exist_use:NT \nicematrix@redefine@check@rerun
151       \ExplSyntaxOff
152     }
153   }
154 }
155 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
156 {
157   \let @@_old_pgful@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`).

```

158   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
159   {
\str_if_eq:ee(TF) is slightly faster than \str_if_eq:nn(TF).
160     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } { 1 } { 3 } }
161     { @@_old_pgful@check@rerun { ##1 } { ##2 } }
162   }
163 }

```

We have to know whether `colortbl` is loaded in particular for the redefinition of `\everycr`. The command `@@_everycr:` will be used only in `@@_some_initialization:`, itself in `\ar@ialign`.

```

164 \hook_gput_code:nnn { begindocument } { . }
165 {
166   \cs_set_protected:Npe @@_everycr:
167   {
168     \IfPackageLoadedTF { colortbl } { CT@everycr \everycr
169     { \noalign { @@_in_everycr: } }
170   }
171   \IfPackageLoadedTF { colortbl }
172   {

```

```

173 \cs_new_eq:NN \@@_old_cellcolor: \cellcolor
174 \cs_new_eq:NN \@@_old_rowcolor: \rowcolor
175 \cs_new_protected:Npn \@@_revert_colortbl:
176 {
177     \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
178     {
179         \cs_set_eq:NN \cellcolor \@@_old_cellcolor:
180         \cs_set_eq:NN \rowcolor \@@_old_rowcolor:
181     }
182 }

```

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).

```

183 \cs_new_protected:Npn \@@_replace_columncolor:
184 {
185     \tl_replace_all:Nnn \g_@@_array_preamble_tl
186     \columncolor
187     \@@_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.

```

188     }
189 }
190 {
191     \cs_new_protected:Npn \@@_revert_colortbl: { }
192     \cs_new_protected:Npn \@@_replace_columncolor:
193     { \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.

```

194 \def \CT@arc@ { }
195 \def \arrayrulecolor #1 # { \CT@arc { #1 } }
196 \def \CT@arc #1 #2
197 {
198     \dim_compare:nNnT \baselineskip = \c_zero_dim { \noalign }
199     { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
200 }

```

Idem for `\CT@drs@`.

```

201 \def \doublerulesepcolor #1 # { \CT@drs { #1 } }
202 \def \CT@drs #1 #2
203 {
204     \dim_compare:nNnT \baselineskip = \c_zero_dim { \noalign }
205     { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
206 }
207 \def \hline
208 {
209     \noalign { \ifnum 0 = ` } \fi
210     \cs_set_eq:NN \hskip \vskip
211     \cs_set_eq:NN \vrule \hrule
212     \cs_set_eq:NN \@width \@height
213     { \CT@arc@ \vline }
214     \futurelet \reserved@a
215     \@xhline
216 }
217 }
218 }

```

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.

```

219 \cs_set_nopar:Npn \@@_standard_cline: #1 { \@@_standard_cline:w #1 \q_stop }
220 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
221 {

```

```

222 \int_if_zero:nT \l_@@_first_col_int { \omit & }
223 \int_compare:nNnT { #1 } > 1
224 { \multispan { \int_eval:n { #1 - 1 } } & }
225 \multispan { \int_eval:n { #2 - #1 + 1 } }
226 {
227 \CT@arc@
228 \leaders \hrule \@height \arrayrulewidth \hfill

```

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

```

229 \skip_horizontal:N \c_zero_dim
230 }

```

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`.

```

231 \everycr { }
232 \cr
233 \noalign { \skip_vertical:n { - \arrayrulewidth } }
234 }

```

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.

```

235 \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`.

```

236 { \@@_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*.

```

237 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
238 \cs_generate_variant:Nn \@@_cline_i:nn { e }
239 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
240 {
241 \tl_if_empty:nTF { #3 }
242 { \@@_cline_iii:w #1|#2-#2 \q_stop }
243 { \@@_cline_ii:w #1|#2-#3 \q_stop }
244 }
245 \cs_set:Npn \@@_cline_ii:w #1|#2-#3- \q_stop
246 { \@@_cline_iii:w #1|#2-#3 \q_stop }
247 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
248 {

```

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).

```

249 \int_compare:nNnT { #1 } < { #2 }
250 { \multispan { \int_eval:n { #2 - #1 } } & }
251 \multispan { \int_eval:n { #3 - #2 + 1 } }
252 {
253 \CT@arc@
254 \leaders \hrule \@height \arrayrulewidth \hfill
255 \skip_horizontal:N \c_zero_dim
256 }

```

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

```

257 \peek_meaning_remove_ignore_spaces:NTF \cline
258 { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
259 { \everycr { } \cr }
260 }

```

¹See question 99041 on TeX StackExchange.

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

```

261 \cs_set:Nn \@@_math_toggle: { $ } % $

262 \cs_new_protected:Npn \@@_set_CTarc:n #1
263 {
264   \tl_if_blank:nF { #1 }
265   {
266     \tl_if_head_eq_meaning:nNTF { #1 } [
267       { \def \CT@arc@ { \color #1 } }
268       { \def \CT@arc@ { \color { #1 } } }
269     ]
270   }
271 \cs_generate_variant:Nn \@@_set_CTarc:n { o }

272 \cs_new_protected:Npn \@@_set_CTdrsc:n #1
273 {
274   \tl_if_head_eq_meaning:nNTF { #1 } [
275     { \def \CT@drsc@ { \color #1 } }
276     { \def \CT@drsc@ { \color { #1 } } }
277   ]

```

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

```

278 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
279 {
280   \tl_if_head_eq_meaning:nNTF { #2 } [
281     { #1 #2 }
282     { #1 { #2 } }
283   ]
284 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```

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

```

285 \cs_new_protected:Npn \@@_color:n #1
286 { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }
287 \cs_generate_variant:Nn \@@_color:n { o }

288 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
289 {
290   \tl_set_rescan:Nno
291     #1
292     {
293       \char_set_catcode_other:N >
294       \char_set_catcode_other:N <
295     }
296   #1
297 }

```

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

```

298 \dim_new:N \l_@@_tmpc_dim
299 \dim_new:N \l_@@_tmpd_dim

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

302 \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.

```
303 \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).

```
304 \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.

```
305 \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.

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

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

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

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

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

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

```
310 \bool_new:N \g_@@_delims_bool
311 \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.

```
312 \bool_new:N \l_@@_preamble_bool
313 \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.

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

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

```
315 \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.

```
316 \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).

```
317 \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.).

```
318 \dim_new:N \l_@@_col_width_dim
319 \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.

```
320 \int_new:N \g_@@_row_total_int
321 \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).

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

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

```
323 \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.

```
324 \tl_new:N \l_@@_hpos_cell_tl
325 \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`.

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

Idem for the mono-row blocks.

```
327 \dim_new:N \g_@@_blocks_ht_dim
328 \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`).

```
329 \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`.

```
330 \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.

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

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

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

```

334 \bool_new:N \l_@@_initial_open_bool
335 \bool_new:N \l_@@_final_open_bool
336 \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.

```

337 \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).

```

338 \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).

```

339 \tl_new:N \l_@@_rule_color_tl

```

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

```

340 \bool_new:N \g_@@_rotate_bool

```

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

```

341 \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.

```

342 \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`).

```

343 \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`.

```

344 \bool_new:N \g_@@_V_of_X_bool

```

```

345 \bool_new:N \g_@@_caption_finished_bool

```

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

```

346 \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 }`).

```

347 \tl_new:N \g_@@_aux_tl

```

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

```

348 \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.

```

349 \seq_new:N \g_@@_size_seq

```

```

350 \tl_new:N \g_@@_left_delim_tl

```

```

351 \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}`).

```
352 \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`).

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

For `\multicolumn`.

```
354 \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`.

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

```
356 \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 `..`.

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

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

```
359 \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`).

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

```
361 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
362 {
```

```
363   \if_mode_math: \else:
```

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

```
365   \fi:
```

```
366 }
```

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.

```
367 \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”.

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

```
369 \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`).

```
370 \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*.

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

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

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

```
374 \bool_new:N \l_@@_notes_no_print_bool
```

`\g_@@_cbic_clist` is for `create-blocks-in-column`

```
375 \clist_new:N \g_@@_cbic_clist
```

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)`.

```
376 \cs_new:Npn \@@_full_name_env:
377 {
378   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
379   { command \space \c_backslash_str \g_@@_name_env_str }
380   { environment \space \{ \g_@@_name_env_str \} }
381 }
```

```
382 \tl_new:N \g_@@_cell_after_hook_tl
```

The argument is given by curryfication.

```
383 \cs_new_protected:Npn \@@_put_in_cell_after_hook:n
384 { \tl_gput_right:Nn \g_@@_cell_after_hook_tl }
```

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

```
385 \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.

```
386 \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.

```
387 \tl_new:N \g_@@_pre_code_before_tl
388 \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.

```
389 \tl_new:N \g_@@_pre_code_after_tl
390 \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.

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

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

```
392 \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.

```
393 \int_new:N \l_@@_old_iRow_int
394 \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).

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

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

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

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

```
397 \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.

```
398 \bool_new:N \l_@@_X_columns_aux_bool
```

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

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

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

```
401 \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).

```
402 \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).

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

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

```
405 \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).

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

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

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

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

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

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

```

411 \dim_new:N \g_@@_dp_row_zero_dim
412 \dim_new:N \g_@@_ht_row_zero_dim
413 \dim_new:N \g_@@_ht_row_one_dim
414 \dim_new:N \g_@@_dp_ante_last_row_dim
415 \dim_new:N \g_@@_ht_last_row_dim
416 \dim_new:N \g_@@_dp_last_row_dim

```

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

```

417 \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”.

```

418 \dim_new:N \g_@@_width_last_col_dim
419 \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.

```

420 \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!).

```

421 \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.

```

422 \seq_new:N \g_@@_future_pos_of_blocks_seq

```

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

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

```

423 \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.

```

424 \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`).

```

425 \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`).

```

426 \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.

```
427 \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).

```
428 \seq_new:N \g_@@_multicolumn_cells_seq
429 \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.

```
430 \int_new:N \g_@@_ddots_int
431 \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.

```
432 \dim_new:N \g_@@_delta_x_one_dim
433 \dim_new:N \g_@@_delta_y_one_dim
434 \dim_new:N \g_@@_delta_x_two_dim
435 \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).

```
436 \int_new:N \l_@@_row_min_int
437 \int_new:N \l_@@_row_max_int
438 \int_new:N \l_@@_col_min_int
439 \int_new:N \l_@@_col_max_int

440 \int_new:N \l_@@_initial_i_int
441 \int_new:N \l_@@_initial_j_int
442 \int_new:N \l_@@_final_i_int
443 \int_new:N \l_@@_final_j_int
```

The following counters will be used when drawing the rules.

```
444 \int_new:N \l_@@_start_int
445 \int_set:Nn \l_@@_start_int 1
446 \int_new:N \l_@@_end_int
447 \int_new:N \l_@@_local_start_int
448 \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.

```
449 \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).

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

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

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

```
451 \tl_new:N \l_@@_fill_tl
452 \tl_new:N \l_@@_opacity_tl
453 \tl_new:N \l_@@_draw_tl
454 \seq_new:N \l_@@_tikz_seq
455 \clist_new:N \l_@@_borders_clist
456 \dim_new:N \l_@@_rounded_corners_dim
```

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

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

```
457 \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`.

```
458 \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.

```
459 \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.

```
460 \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`).

```
461 \str_new:N \l_@@_hpos_block_str
462 \str_set:Nn \l_@@_hpos_block_str { c }
463 \bool_new:N \l_@@_hpos_of_block_cap_bool
464 \bool_new:N \l_@@_p_block_bool
```

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

```
465 \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).

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

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

```
467 \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).

```
468 \bool_new:N \l_@@_vlines_block_bool
469 \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.

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

```

471 \dim_new:N \l_@@_submatrix_extra_height_dim
472 \dim_new:N \l_@@_submatrix_left_xshift_dim
473 \dim_new:N \l_@@_submatrix_right_xshift_dim
474 \clist_new:N \l_@@_hlines_clist
475 \clist_new:N \l_@@_vlines_clist
476 \clist_new:N \l_@@_submatrix_hlines_clist
477 \clist_new:N \l_@@_submatrix_vlines_clist

```

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

```

478 \bool_new:N \l_@@_hlines_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.

```

479 \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`).

```

480 \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.

```

481 \int_new:N \l_@@_first_row_int
482 \int_set:Nn \l_@@_first_row_int 1

```

• 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.

```

483 \int_new:N \l_@@_first_col_int
484 \int_set:Nn \l_@@_first_col_int 1

```

• 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).

```

485 \int_new:N \l_@@_last_row_int
486 \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

```

487 \bool_new:N \l_@@_last_row_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.

Idem for `\l_@@_last_col_without_value_bool`

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

• Last column

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

```
489 \int_new:N \l_@@_last_col_int
490 \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.

```
491 \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`).

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

Some utilities

```
493 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
494 {
```

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

```
495 \def \l_tmpa_tl { #1 }
496 \def \l_tmpb_tl { #2 }
497 }
```

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`.

```
498 \cs_new_protected:Npn \@@_expand_clist_hvlines:NN #1 #2
499 {
500 \clist_if_in:NnF #1 { all }
501 {
502 \clist_clear:N \l_tmpa_clist
503 \clist_map_inline:Nn #1
504 {
505 \tl_if_head_eq_meaning:nNTF { ##1 } -
506 {
```

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.

```

507         \int_if_zero:nF { #2 }
508         {
509             \clist_put_right:Ne \l_tmpa_clist
510             { \int_eval:n { #2 + (##1) + 1 } }
511         }
512     }
513     {

```

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

```

514         \tl_if_in:nnTF { ##1 } { - }
515         { \@@_cut_on_hyphen:w ##1 \q_stop }
516         {

```

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

```

517         \def \l_tmpa_tl { ##1 }
518         \def \l_tmpb_tl { ##1 }
519     }
520     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
521     { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
522 }
523 }
524 \tl_set_eq:NN #1 \l_tmpa_clist
525 }
526 }

```

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.

```

527 \hook_gput_code:nnn { begindocument } { . }
528 {
529     \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
530     \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
531 }

```

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 tabulernote}`. The first component is the optional argument (between square brackets) of the command `\tabulernote` (if the optional argument is not used, the value will be the special marker expressed by `\NoValue`).
- 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 `tabulernote` 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.

```
532 \newcounter { tabulernote }
```

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_@@_tabulernote_int`.

```
533 \int_new:N \g_@@_tabulernote_int
534 \cs_set:Npn \theHtabulernote { \int_use:N \g_@@_tabulernote_int }

535 \seq_new:N \g_@@_notes_seq
536 \seq_new:N \g_@@_notes_in_caption_seq
```

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

```
537 \tl_new:N \g_@@_tabulernote_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.

```
538 \seq_new:N \l_@@_notes_labels_seq
539 \newcounter { nicematrix_draft }
540 \cs_new_protected:Npn \@@_notes_format:n #1
541 {
542   \setcounter { nicematrix_draft } { #1 }
543   \@@_notes_style:n { nicematrix_draft }
544 }
```

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

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

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

```
546 \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`.

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

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

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`.

```
548 \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`).

```
549 \hook_gput_code:nnn { begindocument } { . }
550 {
551   \IfPackageLoadedTF { enumitem }
552   {
```

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.

```
553     \newlist { tabularnotes } { enumerate } { 1 }
554     \setlist [ tabularnotes ]
555     {
556       topsep = \c_zero_dim ,
557       noitemsep ,
558       leftmargin = * ,
559       align = left ,
560       labelsep = \c_zero_dim ,
561       label =
562         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
563     }
564     \newlist { tabularnotes* } { enumerate* } { 1 }
565     \setlist [ tabularnotes* ]
566     {
567       afterlabel = \nobreak ,
568       itemjoin = \quad ,
569       label =
570         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
571     }
```

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`.

```
572 \NewDocumentCommand { \tabularnote } { o m }
573 {
574   \bool_lazy_or:nnT \l_@@_in_env_bool { \cs_if_exist_p:N \@capttype }
575   {
576     \bool_lazy_and:nnTF \l_@@_in_env_bool { ! \l_@@_tabular_bool }
577     { \@@_error:n { tabularnote~forbidden } }
578     {
```

The second argument of `\@@_tabularnote_caption:nn` ou `\@@_tabularnote:nn` is provided by currying.

```
579     \bool_if:NTF \l_@@_in_caption_bool
580     {
581       \tl_if_novalue:nTF { #1 }
582       { \@@_tabularnote_caption:nn { #1 } }
583       { \@@_tabularnote_caption:nn { \exp_not:n { #1 } } }
584     }
585     {
586       \tl_if_novalue:nTF { #1 }
587       { \@@_tabularnote:nn { #1 } }
588       { \@@_tabularnote:nn { \exp_not:n { #1 } } }
589     }
```

```

590             { #2 }
591         }
592     }
593 }
594 }
595 {
596     \NewDocumentCommand \tabularnote { o m }
597     { \@@_err_enumitem_not_loaded: }
598 }
599 }
600 \cs_new_protected:Npn \@@_err_enumitem_not_loaded:
601 {
602     \@@_error_or_warning:n { enumitem-not-loaded }
603     \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
604 }
605 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
606 { \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`.

```

607 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
608 {

```

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).

```

609     \int_zero:N \l_tmpa_int
610     \bool_if:NT \l_@@_notes_detect_duplicates_bool
611     {

```

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`.

```

612     \int_zero:N \l_tmpb_int
613     \seq_map_indexed_inline:Nn \g_@@_notes_seq
614     {
615         \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
616         \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
617         {
618             \tl_if_novalue:nTF { #1 }
619             { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
620             { \int_set:Nn \l_tmpa_int { ##1 } }
621             \seq_map_break:
622         }
623     }
624     \int_if_zero:nF \l_tmpa_int
625     { \int_add:Nn \l_tmpa_int { \g_@@_notes_caption_int } }
626 }
627 \int_if_zero:nT \l_tmpa_int
628 {
629     \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
630     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
631 }
632 \seq_put_right:Ne \l_@@_notes_labels_seq

```

```

633 {
634   \tl_if_novalue:nTF { #1 }
635   {
636     \@@_notes_format:n
637     {
638       \int_eval:n
639       {
640         \int_if_zero:nTF \l_tmpa_int
641         \c@tabularnote
642         \l_tmpa_int
643       }
644     }
645   }
646   { #1 }
647 }
648 \peek_meaning:NF \tabularnote
649 {

```

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`.

```

650   \hbox_set:Nn \l_tmpa_box
651   {

```

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

```

652     \@@_notes_label_in_tabular:n
653     { \seq_use:Nn \l_@@_notes_labels_seq { , } }
654   }

```

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

```

655   \int_gdecr:N \c@tabularnote
656   \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.

```

657   \int_gincr:N \g_@@_tabularnote_int
658   \refstepcounter { tabularnote }
659   \int_compare:nNnT \l_tmpa_int = \c@tabularnote
660   { \int_gincr:N \c@tabularnote }
661   \seq_clear:N \l_@@_notes_labels_seq
662   \bool_lazy_or:nnTF
663   { \str_if_eq_p:ee c \l_@@_hpos_cell_tl }
664   { \str_if_eq_p:ee r \l_@@_hpos_cell_tl }
665   {
666     \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).

```

667     \skip_horizontal:n { \box_wd:N \l_tmpa_box }
668   }
669   { \box_use:N \l_tmpa_box }
670 }
671 }

```

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`.

```

672 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2

```



```

673 {
674   \bool_if:NTF \g_@@_caption_finished_bool
675   {
676     \int_compare:nNt \c@tabulernote = \g_@@_notes_caption_int
677     { \int_gzero:N \c@tabulernote }

```

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`!

```

678     \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
679     { \@@_error:n { Identical~notes~in~caption } }
680   }
681 {

```

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

```

682     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
683     {

```

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 `\tabulernote` in the caption.

```

684         \bool_gset_true:N \g_@@_caption_finished_bool
685         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabulernote
686         \int_gzero:N \c@tabulernote
687     }
688     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
689 }

```

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!

```

690   \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabulernote }
691   \seq_put_right:Ne \l_@@_notes_labels_seq
692   {
693     \tl_if_novalue:nTF { #1 }
694     { \@@_notes_format:n { \int_use:N \c@tabulernote } }
695     { #1 }
696   }
697   \peek_meaning:NF \tabulernote
698   {
699     \@@_notes_label_in_tabular:n { \seq_use:Nn \l_@@_notes_labels_seq { , } }
700     \seq_clear:N \l_@@_notes_labels_seq
701   }
702 }

703 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
704 { \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.

```

705 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
706 {
707   \begin { pgfscope }
708   \pgfset
709   {
710     inner~sep = \c_zero_dim ,
711     minimum~size = \c_zero_dim
712   }
713   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }

```

```

714 \pgfnode
715 { rectangle }
716 { center }
717 {
718   \vbox_to_ht:nn
719   { \dim_abs:n { #5 - #3 } }
720   {
721     \vfill
722     \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
723   }
724 }
725 { #1 }
726 { }
727 \end { pgfscope }
728 }

```

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.

```

729 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
730 {
731   \begin { pgfscope }
732   \pgfset
733   {
734     inner~sep = \c_zero_dim ,
735     minimum~size = \c_zero_dim
736   }
737   \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
738   \pgfpointdiff { #3 } { #2 }
739   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
740   \pgfnode
741   { rectangle }
742   { center }
743   {
744     \vbox_to_ht:nn
745     { \dim_abs:n \l_tmpb_dim }
746     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
747   }
748   { #1 }
749   { }
750   \end { pgfscope }
751 }

```

7 The options

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

```

752 \tl_new:N \l_@@_caption_tl
753 \tl_new:N \l_@@_short_caption_tl
754 \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).

```

755 \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`.

```
756 \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`).

```
757 \dim_new:N \l_@@_cell_space_top_limit_dim
758 \dim_new:N \l_@@_cell_space_bottom_limit_dim
```

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

```
759 \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.

```
760 \dim_new:N \l_@@_xdots_inter_dim
761 \hook_gput_code:nnn { begindocument } { . }
762 { \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).

```
763 \dim_new:N \l_@@_xdots_shorten_start_dim
764 \dim_new:N \l_@@_xdots_shorten_end_dim
765 \hook_gput_code:nnn { begindocument } { . }
766 {
767   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
768   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
769 }
```

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.

```
770 \dim_new:N \l_@@_xdots_radius_dim
771 \hook_gput_code:nnn { begindocument } { . }
772 { \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.

```
773 \tl_new:N \l_@@_xdots_line_style_tl
774 \tl_const:Nn \c_@@_standard_tl { standard }
775 \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`.

```
776 \bool_new:N \l_@@_light_syntax_bool
777 \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).

```
778 \tl_new:N \l_@@_baseline_tl
779 \tl_set:Nn \l_@@_baseline_tl { c }
```

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

```
780 \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`).

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

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

```
782 \bool_new:N \l_@@_parallelize_diags_bool
783 \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.

```
784 \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.).

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

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

```
786 \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).

```
787 \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!

```
788 \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.

```
789 \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”.

```
790 \bool_new:N \l_@@_medium_nodes_bool
791 \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).

```
792 \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`).

```
793 \dim_new:N \l_@@_left_margin_dim
794 \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`.

```
795 \dim_new:N \l_@@_extra_left_margin_dim
796 \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.

```
797 \tl_new:N \l_@@_end_of_row_tl
798 \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 “:”.

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

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

```
800 \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.

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

```
802 \keys_define:nn { nicematrix / xdots }
803 {
804   nullify .bool_set:N = \l_@@_nullify_dots_bool ,
805   nullify .default:n = true ,
806   brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
807   brace-shift .value_required:n = true ,
808   brace-shift+ .code:n =
809     \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
810   brace-shift+ .value_required:n = true ,
811   brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
812   Vbrace .bool_set:N = \l_@@_Vbrace_bool ,
813   shorten-start .code:n =
814     \hook_gput_code:nnn { begindocument } { . }
815     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
816   shorten-start .value_required:n = true ,
817   shorten-start+ .code:n =
818     \hook_gput_code:nnn { begindocument } { . }
819     { \dim_add:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
820   shorten-start+ .meta:n = { shorten-start += #1 } ,
821   shorten-start+ .value_required:n = true ,
822   shorten-end .code:n =
823     \hook_gput_code:nnn { begindocument } { . }
824     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
825   shorten-end .value_required:n = true ,
826   shorten-end+ .code:n =
827     \hook_gput_code:nnn { begindocument } { . }
828     { \dim_add:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
829   shorten-end+ .value_required:n = true ,
830   shorten-end~+ .meta:n = { shorten-end += #1 } ,
831   shorten .code:n =
832     \hook_gput_code:nnn { begindocument } { . }
833     {
834       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
835       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
```

```

836     } ,
837     shorten .value_required:n = true ,
838     shorten+ .code:n =
839     \hook_gput_code:nnn { begindocument } { . }
840     {
841         \dim_add:Nn \l_@@_xdots_shorten_start_dim { #1 }
842         \dim_add:Nn \l_@@_xdots_shorten_end_dim { #1 }
843     } ,
844     shorten~+ .meta:n = { shorten+ = #1 } ,
845     horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
846     horizontal-labels .default:n = true ,
847     horizontal-label .bool_set:N = \l_@@_xdots_h_labels_bool ,
848     horizontal-label .default:n = true ,
849     line-style .code:n =
850     {
851         \bool_lazy_or:nnTF
852         { \cs_if_exist_p:N \tikzpicture }
853         { \str_if_eq_p:nn { #1 } { standard } }
854         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
855         { \@@_error:n { bad-option-for~line-style } }
856     } ,
857     line-style .value_required:n = true ,
858     color .tl_set:N = \l_@@_xdots_color_tl ,
859     color .value_required:n = true ,
860     radius .code:n =
861     \hook_gput_code:nnn { begindocument } { . }
862     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
863     radius .value_required:n = true ,
864     inter .code:n =
865     \hook_gput_code:nnn { begindocument } { . }
866     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
867     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 `^{...}`.

```

868     down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
869     up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
870     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).

```

871     draw-first .code:n = \prg_do_nothing: ,
872     unknown .code:n =
873     \@@_unknown_key:nn { nicematrix / xdots } { Unknown~key~for~xdots }
874 }

```

```

875 \keys_define:nn { nicematrix / rules }
876 {
877     color .tl_set:N = \l_@@_rules_color_tl ,
878     color .value_required:n = true ,
879     width .dim_set:N = \arrayrulewidth ,
880     width .value_required:n = true ,
881     unknown .code:n = \@@_error:n { Unknown~key~for~rules }
882 }

```

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

```

883 \keys_define:nn { nicematrix / Global }
884 {
885     brace-shift .dim_set:N = \l_@@_brace_shift_dim ,

```

```

886 brace-shift .value_required:n = true ,
887 brace-shift+ .code:n =
888   \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
889 brace-shift+ .value_required:n = true ,
890 brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
891 caption-above .code:n = \@@_error_or_warning:n { caption-above~in-env } ,
892 show-cell-names .code:n = \@@_error_or_warning:n { show-cell-names } ,
893 color-inside .code:n = \@@_fatal:n { key~color-inside } ,
894 colortbl-like .code:n = \@@_fatal:n { key~color-inside } ,
895 ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
896 ampersand-in-blocks .default:n = true ,
897 &-in-blocks .meta:n = ampersand-in-blocks ,
898 no-cell-nodes .code:n =
899   \bool_set_true:N \l_@@_no_cell_nodes_bool
900   \cs_set_protected:Npn \@@_node_cell:
901     { \set@color \box_use_drop:N \l_@@_cell_box } ,
902 no-cell-nodes .value_forbidden:n = true ,
903 rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
904 rounded-corners .default:n = 4 pt ,
905 custom-line .code:n = \@@_custom_line:n { #1 } ,
906 rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
907 rules .value_required:n = true ,
908 standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
909 standard-cline .default:n = true ,
910 cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
911 cell-space-top-limit .value_required:n = true ,
912 cell-space-top-limit+ .code:n =
913   \dim_add:Nn \l_@@_cell_space_top_limit_dim { #1 } ,
914 cell-space-top-limit+ .value_required:n = true ,
915 cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
916 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
917 cell-space-bottom-limit .value_required:n = true ,
918 cell-space-bottom-limit+ .code:n =
919   \dim_add:Nn \l_@@_cell_space_bottom_limit_dim { #1 } ,
920 cell-space-bottom-limit+ .value_required:n = true ,
921 cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
922 cell-space-limits .meta:n =
923   {
924     cell-space-top-limit = #1 ,
925     cell-space-bottom-limit = #1 ,
926   } ,
927 cell-space-limits .value_required:n = true ,
928 cell-space-limits+ .meta:n =
929   {
930     cell-space-top-limit += #1 ,
931     cell-space-bottom-limit += #1 ,
932   } ,
933 cell-space-limits+ .value_required:n = true ,
934 cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
935 xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
936 light-syntax .code:n =
937   \bool_set_true:N \l_@@_light_syntax_bool
938   \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
939 light-syntax .value_forbidden:n = true ,
940 light-syntax-expanded .code:n =
941   \bool_set_true:N \l_@@_light_syntax_bool
942   \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
943 light-syntax-expanded .value_forbidden:n = true ,
944 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
945 end-of-row .value_required:n = true ,
946
947 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
948 first-row .code:n = \int_zero:N \l_@@_first_row_int ,

```

```

949 last-row .int_set:N = \l_@@_last_row_int ,
950 last-row .default:n = -1 ,
951
952 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
953 code-for-first-col .value_required:n = true ,
954 code-for-first-col+ .code:n =
955   { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
956 code-for-first-col+ .value_required:n = true ,
957 code-for-first-col~+ .meta:n = { code-for-first-col+ = #1 } ,
958
959 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
960 code-for-last-col .value_required:n = true ,
961 code-for-last-col+ .code:n =
962   { \tl_put_right:Nn \l_@@_code_for_last_col_tl { #1 } } ,
963 code-for-last-col+ .value_required:n = true ,
964 code-for-last-col~+ .meta:n = { code-for-last-col+ = #1 } ,
965
966 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
967 code-for-first-row .value_required:n = true ,
968 code-for-first-row+ .code:n =
969   { \tl_put_right:Nn \l_@@_code_for_first_row_tl { #1 } } ,
970 code-for-first-row+ .value_required:n = true ,
971 code-for-first-row~+ .meta:n = { code-for-first-row+ = #1 } ,
972
973 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
974 code-for-last-row .value_required:n = true ,
975 code-for-last-row+ .code:n =
976   { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
977 code-for-last-row+ .value_required:n = true ,
978 code-for-last-row~+ .meta:n = { code-for-last-row+ = #1 } ,
979
980 hlines .clist_set:N = \l_@@_hlines_clist ,
981 vlines .clist_set:N = \l_@@_vlines_clist ,
982 hlines .default:n = all ,
983 vlines .default:n = all ,
984 vlines-in-sub-matrix .code:n =
985   {
986     \tl_if_single_token:nTF { #1 }
987     {
988       \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
989       { \@@_error:nn { Forbidden~letter } { #1 } }

```

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

```

990     { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
991   }
992   { \@@_error:n { One~letter~allowed } }
993 } ,
994 vlines-in-sub-matrix .value_required:n = true ,
995 hvlines .code:n =
996   {
997     \bool_set_true:N \l_@@_hvlines_bool
998     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
999     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
1000   } ,
1001 hvlines .value_forbidden:n = true ,
1002 hvlines-except-borders .code:n =
1003   {
1004     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
1005     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
1006     \bool_set_true:N \l_@@_hvlines_bool
1007     \bool_set_true:N \l_@@_except_borders_bool
1008   } ,
1009 hvlines-except-borders .value_forbidden:n = true ,
1010 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.

```

1011   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
1012   renew-dots .value_forbidden:n = true ,
1013   nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
1014   create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
1015   create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
1016   create-extra-nodes .meta:n =
1017     { create-medium-nodes , create-large-nodes } ,
1018   left-margin .dim_set:N = \l_@@_left_margin_dim ,
1019   left-margin .default:n = \arraycolsep ,
1020   right-margin .dim_set:N = \l_@@_right_margin_dim ,
1021   right-margin .default:n = \arraycolsep ,
1022   margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
1023   margin .default:n = \arraycolsep ,
1024   extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
1025   extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
1026   extra-margin .meta:n =
1027     { extra-left-margin = #1 , extra-right-margin = #1 } ,
1028   extra-margin .value_required:n = true ,
1029   respect-arraystretch .code:n =
1030     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
1031   respect-arraystretch .value_forbidden:n = true ,
1032   pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
1033   pgf-node-code .value_required:n = true
1034 }

```

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

```

1035 \keys_define:nn { nicematrix / environments }
1036 {
1037   create-blocks-in-col .code:n = \@@_create_blocks_in_col:n { #1 } ,
1038   create-blocks-in-col .value_required:n = true ,
1039   corners .clist_set:N = \l_@@_corners_clist ,
1040   corners .default:n = { NW , SW , NE , SE } ,
1041   code-before .code:n =
1042     {
1043       \tl_if_empty:nF { #1 }
1044       {
1045         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1046         \bool_set_true:N \l_@@_code_before_bool
1047       }
1048     } ,
1049   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}`.

```

1050   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
1051   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
1052   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
1053   baseline .tl_set:N = \l_@@_baseline_tl ,
1054   baseline .value_required:n = true ,
1055   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)`.

```

1056   \str_if_eq:eeTF { #1 } { auto }
1057   { \bool_set_true:N \l_@@_auto_columns_width_bool }
1058   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
1059   columns-width .value_required:n = true ,
1060   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.

```

1061 \legacy_if:nF { measuring@ }
1062 {
1063     \str_set:Nc \l_@@_name_str { #1 }
1064     \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
1065     { \@@_err_duplicate_names:n { #1 } }
1066     { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
1067 } ,
1068 name .value_required:n = true ,
1069 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
1070 code-after .value_required:n = true ,
1071 }

1072 \cs_set:Npn \@@_err_duplicate_names:n #1
1073 { \@@_error:nn { Duplicate-name } { #1 } }

1074 \keys_define:nn { nicematrix / notes }
1075 {
1076     no-print .bool_set:N = \l_@@_notes_no_print_bool ,
1077     no-print .default:n = true ,
1078     para .bool_set:N = \l_@@_notes_para_bool ,
1079     para .default:n = true ,
1080     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
1081     code-before .value_required:n = true ,
1082     code-before+ .code:n =
1083     \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1084     code-before+ .value_required:n = true ,
1085     code-before~+ .code:n =
1086     \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1087     code-before~+ .value_required:n = true ,
1088     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
1089     code-after .value_required:n = true ,
1090     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
1091     bottomrule .default:n = true ,
1092     style .cs_set:Np = \@@_notes_style:n #1 ,
1093     style .value_required:n = true ,
1094     label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
1095     label-in-tabular .value_required:n = true ,
1096     label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1097     label-in-list .value_required:n = true ,
1098     enumitem-keys .code:n =
1099     {
1100         \hook_gput_code:nnn { begindocument } { . }
1101         {
1102             \IfPackageLoadedT { enumitem }
1103             { \setlist* [ tabularnotes ] { #1 } }
1104         }
1105     } ,
1106     enumitem-keys .value_required:n = true ,
1107     enumitem-keys-para .code:n =
1108     {
1109         \hook_gput_code:nnn { begindocument } { . }
1110         {
1111             \IfPackageLoadedT { enumitem }
1112             { \setlist* [ tabularnotes* ] { #1 } }
1113         }
1114     } ,
1115     enumitem-keys-para .value_required:n = true ,
1116     detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1117     detect-duplicates .default:n = true ,
1118     unknown .code:n =
1119     \@@_unknown_key:nn { nicematrix / notes } { Unknown~key~for~notes }
1120 }

```

```

1121 \keys_define:nn { nicematrix / delimiters }
1122 {
1123   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1124   max-width .default:n = true ,
1125   color .tl_set:N = \l_@@_delimiters_color_tl ,
1126   color .value_required:n = true ,
1127 }

```

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

```

1128 \keys_define:nn { nicematrix }
1129 {
1130   NiceMatrixOptions .inherit:n =
1131     { nicematrix / Global } ,
1132   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1133   NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1134   NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1135   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1136   SubMatrix / rules .inherit:n = nicematrix / rules ,
1137   CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1138   CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1139   CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1140   NiceMatrix .inherit:n =
1141     {
1142       nicematrix / Global ,
1143       nicematrix / environments ,
1144     } ,
1145   NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1146   NiceMatrix / rules .inherit:n = nicematrix / rules ,
1147   NiceTabular .inherit:n =
1148     {
1149       nicematrix / Global ,
1150       nicematrix / environments
1151     } ,
1152   NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1153   NiceTabular / rules .inherit:n = nicematrix / rules ,
1154   NiceTabular / notes .inherit:n = nicematrix / notes ,
1155   NiceArray .inherit:n =
1156     {
1157       nicematrix / Global ,
1158       nicematrix / environments ,
1159     } ,
1160   NiceArray / xdots .inherit:n = nicematrix / xdots ,
1161   NiceArray / rules .inherit:n = nicematrix / rules ,
1162   pNiceArray .inherit:n =
1163     {
1164       nicematrix / Global ,
1165       nicematrix / environments ,
1166     } ,
1167   pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1168   pNiceArray / rules .inherit:n = nicematrix / rules
1169 }

```

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

```

1170 \keys_define:nn { nicematrix / NiceMatrixOptions }
1171 {
1172   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1173   delimiters / color .value_required:n = true ,
1174   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1175   delimiters / max-width .default:n = true ,
1176   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,

```

```

1177     delimiters .value_required:n = true ,
1178     width .dim_set:N = \l_@@_width_dim ,
1179     width .value_required:n = true ,
1180     last-col .code:n =
1181       \tl_if_empty:nF { #1 }
1182       { \@@_error:n { last-col~non-empty~for~NiceMatrixOptions } }
1183       \int_zero:N \l_@@_last_col_int ,
1184     small .bool_set:N = \l_@@_small_bool ,
1185     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.

```

1186     renew-matrix .code:n = \@@_renew_matrix: ,
1187     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}`.

```

1188     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.

```

1189     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)`.

```

1190     \str_if_eq:eeTF { #1 } { auto }
1191     { \@@_error:n { Option~auto~for~columns-width } }
1192     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1193     allow-duplicate-names .code:n =
1194     \cs_set:Nn \@@_err_duplicate_names:n { } ,
1195     allow-duplicate-names .value_forbidden:n = true ,
1196     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1197     notes .value_required:n = true ,
1198     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1199     sub-matrix .value_required:n = true ,
1200     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1201     matrix / columns-type .value_required:n = true ,
1202     caption-above .bool_set:N = \l_@@_caption_above_bool ,
1203     caption-above .default:n = true ,
1204     unknown .code:n =
1205     \@@_unknown_key:nn
1206     { nicematrix / Global , nicematrix / NiceMatrixOptions }
1207     { Unknown-key-for-NiceMatrixOptions }
1208 }

```

`\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.

```

1209 \NewDocumentCommand \NiceMatrixOptions { m }
1210 { \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.

```

1211 \keys_define:nn { nicematrix / NiceMatrix }
1212 {
1213     last-col .code:n = \tl_if_empty:nTF { #1 }
1214     {

```

```

1215         \bool_set_true:N \l_@@_last_col_without_value_bool
1216         \int_set:Nn \l_@@_last_col_int { -1 }
1217     }
1218     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1219     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1220     columns-type .value_required:n = true ,
1221     l .meta:n = { columns-type = l } ,
1222     r .meta:n = { columns-type = r } ,
1223     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1224     delimiters / color .value_required:n = true ,
1225     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1226     delimiters / max-width .default:n = true ,
1227     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1228     delimiters .value_required:n = true ,
1229     small .bool_set:N = \l_@@_small_bool ,
1230     small .value_forbidden:n = true ,
1231     unknown .code:n =
1232         \@@_unknown_key:nn
1233         { nicematrix / Global , nicematrix / environments , nicematrix / NiceMatrix }
1234         { Unknown-key-for-NiceMatrix }
1235 }

```

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

```

1236 \keys_define:nn { nicematrix / NiceArray }
1237 {

```

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.

```

1238     small .bool_set:N = \l_@@_small_bool ,
1239     small .value_forbidden:n = true ,
1240     last-col .code:n = \tl_if_empty:nF { #1 }
1241         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1242         \int_zero:N \l_@@_last_col_int ,
1243     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1244     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1245     unknown .code:n =
1246         \@@_unknown_key:nn
1247         { nicematrix / NiceArray , nicematrix / Global , nicematrix / environments }
1248         { Unknown-key-for-NiceArray }
1249 }
1250 \keys_define:nn { nicematrix / pNiceArray }
1251 {
1252     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1253     last-col .code:n = \tl_if_empty:nF { #1 }
1254         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1255         \int_zero:N \l_@@_last_col_int ,
1256     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1257     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1258     delimiters / color .value_required:n = true ,
1259     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1260     delimiters / max-width .default:n = true ,
1261     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1262     delimiters .value_required:n = true ,
1263     small .bool_set:N = \l_@@_small_bool ,
1264     small .value_forbidden:n = true ,
1265     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1266     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1267     unknown .code:n =
1268         \@@_unknown_key:nn
1269         { nicematrix / pNiceArray , nicematrix / Global , nicematrix / environments }
1270         { Unknown-key-for-NiceMatrix }
1271 }

```

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

```
1272 \keys_define:nn { nicematrix / NiceTabular }
1273 {
```

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.

```
1274   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1275           \bool_set_true:N \l_@@_width_used_bool ,
1276   width .value_required:n = true ,
1277   notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1278   tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1279   tabularnote .value_required:n = true ,
1280   caption .tl_set:N = \l_@@_caption_tl ,
1281   caption .value_required:n = true ,
1282   short-caption .tl_set:N = \l_@@_short_caption_tl ,
1283   short-caption .value_required:n = true ,
1284   label .tl_set:N = \l_@@_label_tl ,
1285   label .value_required:n = true ,
1286   last-col .code:n = \tl_if_empty:nF { #1 }
1287           { \@@_error:n { last-col-non-empty-for-NiceArray } }
1288           \int_zero:N \l_@@_last_col_int ,
1289   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1290   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1291   unknown .code:n =
1292     \@@_unknown_key:nn
1293     { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
1294     { Unknown-key-for-NiceTabular }
1295 }
```

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
```

```
1296 \keys_define:nn { nicematrix / CodeAfter }
1297 {
1298   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1299   delimiters / color .value_required:n = true ,
1300   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1301   rules .value_required:n = true ,
1302   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1303   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1304   sub-matrix .value_required:n = true ,
1305   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1306 }
```

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}`).

```
1307 \cs_new_protected:Npn \@@_cell_begin:
1308 {
```

`\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.

```
1309 \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).

```
1310 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1311 \cs_set_eq:NN \Hline \@@_Hline_in_cell:
```

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

```
1312 \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.

Here is a version with the standard syntax of L3.

```
\int_compare:nNtT { \c@jCol } = { 1 }
{ \int_compare:nNtT \l_@@_first_col_int = { 1 } { \@@_begin_of_row: } }
```

We will use a version a little more efficient.

```
1313 \if_int_compare:w \c@jCol = \c_one_int
1314 \if_int_compare:w \l_@@_first_col_int = \c_one_int
1315 \@@_begin_of_row:
1316 \fi:
1317 \fi:
```

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

```
1318 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1319 \@@_tuning_not_tabular_begin:
1320 \@@_tuning_exterior_rows:
1321 \g_@@_row_style_tl
1322 }
1323 \cs_new_protected:Npn \@@_tuning_exterior_rows: { }
```

Here is a version with the standard syntax of L3.

```
\cs_new_protected:Npn \@@_tuning_first_last_row:
{
  \int_if_zero:nTF { \c@iRow }
  {
    \int_if_zero:nF { \c@jCol }
    {
      \l_@@_code_for_first_row_tl
      \xglobal \colorlet { nicematrix-first-row } { . }
    }
  }
  { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row: }
}
```

We will use a version a little more efficient.

```
1324 \cs_new_protected:Npn \@@_tuning_first_last_row:
1325 {
1326   \if_int_compare:w \c@iRow = \c_zero_int
1327   \if_int_compare:w \c@jCol > \c_zero_int
1328   \l_@@_code_for_first_row_tl
1329   \xglobal \colorlet { nicematrix-first-row } { . }
1330   \fi:
1331   \else:
1332   \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row:
1333   \fi:
1334 }
```

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.

```
1335 \cs_new_protected:Npn \@@_tuning_last_row:
1336 {
1337   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1338     \l_@@_code_for_last_row_tl
1339     \xglobal \colorlet { nicematrix-last-row } { . }
1340   \fi:
1341 }
```

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

```
1342 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```
1343 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1344 {
1345   \m@th
1346   $ % $
```

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

```
1347 \@@_tuning_key_small:
1348 }
1349 \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.

```
1350 \cs_new_protected:Npn \@@_begin_of_row:
1351 {
1352   \int_gincr:N \c@iRow
1353   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1354   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1355   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1356   \pgfpicture
1357   \pgfrememberpicturepositiononpagetrue
1358   \pgfcoordinate
1359     { \@@_env: - row - \int_use:N \c@iRow - base }
1360     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1361   \str_if_empty:NF \l_@@_name_str
1362   {
1363     \pgfnodealias
1364       { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1365       { \@@_env: - row - \int_use:N \c@iRow - base }
1366   }
1367   \endpgfpicture
1368 }
```

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. Here is a version with the standard syntax of L3.


```

\cs_new_protected:Npn \@@_update_for_first_and_last_row:
{
  \int_if_zero:nTF { \c@iRow }
  {
    \dim_compare:nNnT
    { \box_dp:N \l_@@_cell_box } > { \g_@@_dp_row_zero_dim }
    { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
    \dim_compare:nNnT
    { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_zero_dim }
    { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
  }
  {
    \int_compare:nNnT { \c@iRow } = { 1 }
    {
      \dim_compare:nNnT
      { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_one_dim }
      { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
    }
  }
}

```

We will use a version a little more efficient.

```

1369 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1370 {
1371   \if_int_compare:w \c@iRow = \c_zero_int
1372     \if_dim:w \box_dp:N \l_@@_cell_box > \g_@@_dp_row_zero_dim
1373       \global \g_@@_dp_row_zero_dim = \box_dp:N \l_@@_cell_box
1374     \fi:
1375     \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_zero_dim
1376       \global \g_@@_ht_row_zero_dim = \box_ht:N \l_@@_cell_box
1377     \fi:
1378   \else:
1379     \if_int_compare:w \c@iRow = \c_one_int
1380       \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_one_dim
1381         \global \g_@@_ht_row_one_dim = \box_ht:N \l_@@_cell_box
1382       \fi:
1383     \fi:
1384   \fi:
1385 }

1386 \cs_new_protected:Npn \@@_rotate_cell_box:
1387 {
1388   \box_rotate:Nn \l_@@_cell_box { 90 }
1389   \bool_if:NTF \g_@@_rotate_c_bool
1390   {
1391     \hbox_set:Nn \l_@@_cell_box
1392     {
1393       \m@th
1394       $ % $
1395       \vcenter { \box_use:N \l_@@_cell_box }
1396       $ % $
1397     }
1398   }
1399   {
1400     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1401     {
1402       \vbox_set_top:Nn \l_@@_cell_box
1403       {
1404         \vbox_to_zero:n { }
1405         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1406         \box_use:N \l_@@_cell_box
1407       }
1408     }
1409   }

```

```

1409     }
1410     \bool_gset_false:N \g_@@_rotate_bool
1411     \bool_gset_false:N \g_@@_rotate_c_bool
1412 }

```

Here is a version of the command \@@_adjust_size_box: with the syntax of standard L3.

```

\cs_new_protected:Npn \@@_adjust_size_box:
{
  \dim_compare:nNnT { \g_@@_blocks_wd_dim } > { \c_zero_dim }
  {
    \dim_compare:nNnT \g_@@_blocks_wd_dim > { \box_wd:N \l_@@_cell_box }
    { \box_set_wd:Nn \l_@@_cell_box \g_@@_blocks_wd_dim }
    \dim_gzero:N \g_@@_blocks_wd_dim
  }
  \dim_compare:nNnT { \g_@@_blocks_dp_dim } > { \c_zero_dim }
  {
    \dim_compare:nNnT \g_@@_blocks_dp_dim > { \box_dp:N \l_@@_cell_box }
    { \box_set_dp:Nn \l_@@_cell_box \g_@@_blocks_dp_dim }
    \dim_gzero:N \g_@@_blocks_dp_dim
  }
  \dim_compare:nNnT { \g_@@_blocks_ht_dim } > { \c_zero_dim }
  {
    \dim_compare:nNnT \g_@@_blocks_ht_dim > { \box_ht:N \l_@@_cell_box }
    { \box_set_ht:Nn \l_@@_cell_box \g_@@_blocks_ht_dim }
    \dim_gzero:N \g_@@_blocks_ht_dim
  }
}

```

Here is a version slightly more efficient.

```

1413 \cs_set_protected:Npn \@@_adjust_size_box:
1414 {
1415   \if_dim:w \g_@@_blocks_wd_dim > \c_zero_dim
1416     \if_dim:w \g_@@_blocks_wd_dim > \box_wd:N \l_@@_cell_box
1417       \box_wd:N \l_@@_cell_box = \g_@@_blocks_wd_dim
1418     \fi:
1419     \global \g_@@_blocks_wd_dim = \c_zero_dim
1420   \fi:
1421   \if_dim:w \g_@@_blocks_dp_dim > \c_zero_dim
1422     \if_dim:w \g_@@_blocks_dp_dim > \box_dp:N \l_@@_cell_box
1423       \box_dp:N \l_@@_cell_box = \g_@@_blocks_dp_dim
1424     \fi
1425     \global \g_@@_blocks_dp_dim = \c_zero_dim
1426   \fi:
1427   \if_dim:w \g_@@_blocks_ht_dim > \c_zero_dim
1428     \if_dim:w \g_@@_blocks_ht_dim > \box_ht:N \l_@@_cell_box
1429       \box_ht:N \l_@@_cell_box = \g_@@_blocks_ht_dim
1430     \fi:
1431     \global \g_@@_blocks_ht_dim = \c_zero_dim
1432   \fi:
1433 }
1434 \cs_new_protected:Npn \@@_cell_end:
1435 {

```

The following command is nullified in the tabulars.

```

1436   \@@_tuning_not_tabular_end:
1437   \hbox_set_end:
1438   \@@_cell_end_i:
1439 }

```

```

\cs_new_protected:Npn \@@_cell_end_i:
{
  \g_@@_cell_after_hook_tl

```

```

\bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
\@@_adjust_size_box:
\box_set_ht:Nn \l_@@_cell_box
{ \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
\box_set_dp:Nn \l_@@_cell_box
{ \box_dp:N \l_@@_cell_box + \l_@@_cell_space_bottom_limit_dim }
\@@_update_max_cell_width:
\@@_update_for_first_and_last_row:
\bool_if:NTF \g_@@_empty_cell_bool
{ \box_use_drop:N \l_@@_cell_box }
{
  \bool_if:NTF \g_@@_not_empty_cell_bool
  { \@@_print_node_cell: }
  {
    \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
    { \@@_print_node_cell: }
    { \box_use_drop:N \l_@@_cell_box }
  }
}
\int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
{ \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
\bool_gset_false:N \g_@@_empty_cell_bool
\bool_gset_false:N \g_@@_not_empty_cell_bool
}

```

Here is a version slightly more efficient.

```

1440 \cs_new_protected:Npn \@@_cell_end_i:
1441 {

```

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.

```

1442   \g_@@_cell_after_hook_tl
1443   \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
1444   \@@_adjust_size_box:
1445   \box_set_ht:Nn \l_@@_cell_box
1446   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1447   \box_set_dp:Nn \l_@@_cell_box
1448   { \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”).

```

1449   \@@_update_max_cell_width:

```

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

```

1450   \@@_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.

```

1451 \bool_if:NTF \g_@@_empty_cell_bool
1452 { \box_use_drop:N \l_@@_cell_box }
1453 {
1454   \bool_if:NTF \g_@@_not_empty_cell_bool
1455   \@@_print_node_cell:
1456   {
1457     \if_dim:w \box_wd:N \l_@@_cell_box > \c_zero_dim
1458     \@@_print_node_cell:
1459     \else:
1460     \box_use_drop:N \l_@@_cell_box
1461     \fi:
1462   }
1463 }
1464 \if_int_compare:w \c_jCol > \g_@@_col_total_int
1465 \global \g_@@_col_total_int = \c_jCol
1466 \fi:
1467 \global \let \g_@@_empty_cell_bool \c_false_bool
1468 \global \let \g_@@_not_empty_cell_bool \c_false_bool
1469 }

```

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

```

\cs_new_protected:Npn \@@_update_max_cell_width:
{
  \dim_gset:Nn \g_@@_max_cell_width_dim
  { \dim_max:nn { \g_@@_max_cell_width_dim } { \box_wd:N \l_@@_cell_box } }
}

```

We will use the following version, slightly more efficient:

```

1470 \cs_new_protected:Npn \@@_update_max_cell_width:
1471 {
1472   \if_dim:w \box_wd:N \l_@@_cell_box > \g_@@_max_cell_width_dim
1473   \global \g_@@_max_cell_width_dim = \box_wd:N \l_@@_cell_box
1474   \fi:
1475 }

```

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`).

```

1476 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1477 {
1478   \@@_math_toggle:
1479   \hbox_set_end:
1480   \bool_if:NF \g_@@_rotate_bool
1481   {
1482     \hbox_set:Nn \l_@@_cell_box
1483     {
1484       \makebox [ \l_@@_col_width_dim ] [ s ]
1485       { \hbox_unpack_drop:N \l_@@_cell_box }
1486     }
1487   }
1488   \@@_cell_end_i:
1489 }

1490 \pgfset
1491 {

```

```

1492     nicematrix / cell-node /.style =
1493     {
1494         inner~sep = \c_zero_dim ,
1495         minimum~width = \c_zero_dim
1496     }
1497 }

```

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.

```

1498 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1499 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1500 {
1501     \use:c
1502     {
1503         __siunitx_table_align_
1504         \bool_if:NTF \l__siunitx_table_text_bool
1505         \l__siunitx_table_align_text_tl
1506         \l__siunitx_table_align_number_tl
1507         :n
1508     }
1509     { #1 }
1510 }

```

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!

```

1511 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1512 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1513 {
1514     \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1515     \hbox:n
1516     {
1517         \pgfsys@markposition
1518         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1519     }
1520     #1
1521     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1522     \hbox:n
1523     {
1524         \pgfsys@markposition
1525         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1526     }
1527 }

1528 \cs_new_protected:Npn \@@_print_node_cell:
1529 {
1530     \socket_use:nn { nicematrix / siunitx-wrap }
1531     { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1532 }

```

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

```

1533 \cs_new_protected:Npn \@@_node_cell:
1534 {
1535     \pgfpicture
1536     \pgfsetbaseline \c_zero_dim
1537     \pgfrememberpicturepositiononpagetrue
1538     \pgfset { nicematrix / cell-node }

```

```

1539 \pgfnode
1540 { rectangle }
1541 { base }
1542 {

```

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).

```

1543 \sys_if_engine_xetex:T { \set@color }
1544 \box_use:N \l_@@_cell_box
1545 }
1546 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1547 { \l_@@_pgf_node_code_tl }
1548 \str_if_empty:NF \l_@@_name_str
1549 {
1550 \pgfnodealias
1551 { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1552 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1553 }
1554 \endpgfpicture
1555 }

```

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).

```

1556 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1557 {
1558 \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce
1559 { g_@@_ #2 _ lines _ tl }
1560 {
1561 \use:c { @@ _ draw _ #2 : nnn }
1562 { \int_use:N \c@iRow }
1563 { \int_use:N \c@jCol }
1564 { \exp_not:n { #3 } }
1565 }
1566 }

1567 \cs_new_protected:Npn \@@_array:n
1568 {
1569 \dim_set:Nn \col@sep
1570 { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1571 \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1572 { \def \@halignto { } }
1573 { \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`.

```

1574 \@tabarray

```

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

```

1575   [ \str_if_eq:eeTF \l_@@_baseline_tl { c } { c } { t } ]
1576   }
1577 \cs_generate_variant:Nn \@@_array:n { o }

```

We keep in memory the standard version of `\ar@ialign` because we will redefine `\ar@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`.

```

1578 \bool_if:NTF \c_@@_revtex_bool
1579 { \cs_new_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`.

```

1580 { \cs_new_eq:cN { \@@_old_ar@ialign: } \ar@ialign }

```

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

```

1581 \cs_new_protected:Npn \@@_create_row_node:
1582 {
1583   \int_compare:nNt \c@iRow > \g_@@_last_row_node_int
1584   {
1585     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1586     \@@_create_row_node_i:
1587   }
1588 }
1589 \cs_new_protected:Npn \@@_create_row_node_i:
1590 {

```

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

```

1591   \hbox
1592   {
1593     \bool_if:NT \l_@@_code_before_bool
1594     {
1595       \vtop
1596       {
1597         \skip_vertical:N 0.5\arrayrulewidth
1598         \pgfsys@markposition
1599         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1600         \skip_vertical:N -0.5\arrayrulewidth
1601       }
1602     }
1603     \pgfpicture
1604     \pgfrememberpicturepositiononpagetrue
1605     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1606     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1607     \str_if_empty:NF \l_@@_name_str
1608     {
1609       \pgfnodealias
1610       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1611       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1612     }
1613     \endpgfpicture
1614   }
1615 }
1616 \cs_new_protected:Npn \@@_in_everycr:
1617 {
1618   \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1619   \tbl_update_cell_data_for_next_row:
1620   \int_gzero:N \c@jCol

```

```

1621 \bool_gset_false:N \g_@@_after_col_zero_bool
1622 \bool_if:NF \g_@@_row_of_col_done_bool
1623 {
1624 \@@_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).

```

1625 \clist_if_empty:NF \l_@@_hlines_clist
1626 {
1627 \str_if_eq:eeF \l_@@_hlines_clist { all }
1628 {
1629 \clist_if_in:NiT
1630 \l_@@_hlines_clist
1631 { \int_eval:n { \c@iRow + 1 } }
1632 }
1633 {

```

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.

```

1634 \int_compare:nNnT \c@iRow > { -1 }
1635 {
1636 \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1637 { \hrule height \arrayrulewidth width \c_zero_dim }
1638 }
1639 }
1640 }
1641 }
1642 }

```

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

```

1643 \cs_set_protected:Npn \@@_renew_dots:
1644 {
1645 \cs_set_eq:NN \ldots \@@_Ldots:
1646 \cs_set_eq:NN \cdots \@@_Cdots:
1647 \cs_set_eq:NN \vdots \@@_Vdots:
1648 \cs_set_eq:NN \ddots \@@_Ddots:
1649 \cs_set_eq:NN \iddots \@@_Iddots:
1650 \cs_set_eq:NN \dots \@@_Ldots:
1651 \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1652 }

```

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.

```

1653 \hook_gput_code:nnn { begindocument } { . }
1654 {
1655 \IfPackageLoadedTF { booktabs }
1656 {
1657 \cs_new_protected:Npn \@@_patch_booktabs:
1658 { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1659 }
1660 { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1661 }

```

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`.

```

1662 \cs_new_protected:Npn \@@_some_initialization:
1663 {
1664   \@@_everycr:
1665   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1666   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1667   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1668   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1669   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1670   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1671 }

```

`\@@_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`.

```

1672 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1673 {

```

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

```

1674   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1675   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1676   \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.

```

1677   \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.

```

1678   \fp_gzero:N \g_@@_total_X_weight_fp
1679   \bool_gset_false:N \g_@@_V_of_X_bool
1680   \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1681   \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol
1682   \@@_patch_booktabs:
1683   \box_clear_new:N \l_@@_cell_box
1684   \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}`).

```

1685   \bool_if:NT \l_@@_small_bool
1686   {

```

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

```

1687     \def \arraystretch { 0.47 }
1688     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1689     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1690 }

```

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!

```

1691     \bool_if:NT \g_@@_create_cell_nodes_bool
1692     {
1693         \tl_put_right:Nn \@@_begin_of_row:
1694         {
1695             \pgfsys@markposition
1696             { \@@_env: - row - \int_use:N \c@iRow - base }
1697         }
1698         \socket_assign_plug:n { nicematrix / create-cell-nodes } { active }
1699     }

```

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`.

```

1700     \bool_if:NF \c_@@_revtex_bool
1701     {
1702         \def \ar@ialign
1703         {
1704             \tbl_init_cell_data_for_table:
1705             \@@_some_initialization:
1706             \dim_zero:N \tabskip

```

After its first use, the definition of `\ar@ialign` will revert automatically to its default definition. With that 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`.

```

1707             \cs_set_eq:Nc \ar@ialign { \@@_old_ar@ialign: }
1708             \halign
1709         }
1710     }

```

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).

```

1711     \bool_if:NT \c_@@_revtex_bool
1712     {
1713         \IfPackageLoadedT { colortbl }
1714         { \cs_set_protected:Npn \CT@setup { } }
1715     }

```

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).

```

1716     \cs_set_eq:NN \@@_old_ldots: \ldots
1717     \cs_set_eq:NN \@@_old_cdots: \cdots
1718     \cs_set_eq:NN \@@_old_vdots: \vdots
1719     \cs_set_eq:NN \@@_old_ddots: \ddots
1720     \cs_set_eq:NN \@@_old_iddots: \iddots
1721     \bool_if:NTF \l_@@_standard_cline_bool
1722     { \cs_set_eq:NN \cline \@@_standard_cline: }
1723     { \cs_set_eq:NN \cline \@@_cline: }
1724     \cs_set_eq:NN \Ldots \@@_Ldots:

```

```

1725 \cs_set_eq:NN \Cdots \@@_Cdots:
1726 \cs_set_eq:NN \Vdots \@@_Vdots:
1727 \cs_set_eq:NN \Ddots \@@_Ddots:
1728 \cs_set_eq:NN \Iddots \@@_Iddots:
1729 \cs_set_eq:NN \Hline \@@_Hline:
1730 \cs_set_eq:NN \Hspace \@@_Hspace:
1731 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1732 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1733 \cs_set_eq:NN \Block \@@_Block:
1734 \cs_set_eq:NN \rotate \@@_rotate:
1735 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1736 \cs_set_eq:NN \dotfill \@@_dotfill:
1737 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1738 \cs_if_free:NT \Body { \cs_set_eq:NN \Body \@@_Body: }
1739 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1740 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1741 \cs_set_eq:NN \TopRule \@@_TopRule
1742 \cs_set_eq:NN \MidRule \@@_MidRule
1743 \cs_set_eq:NN \BottomRule \@@_BottomRule
1744 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1745 \cs_set_eq:NN \Hbrace \@@_Hbrace
1746 \cs_set_eq:NN \Vbrace \@@_Vbrace
1747 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1748   { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1749 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1750 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1751 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1752 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1753 \int_if_zero:nTF \l_@@_first_row_int
1754   { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_first_last_row: }
1755   {
1756     \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1757     { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row: }
1758   }
1759 \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:`.

```

1760 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1761 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1762   { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1763 \@@_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!).

```

1764 \tl_if_exist:NT \l_@@_note_in_caption_tl
1765   {
1766     \tl_if_empty:NF \l_@@_note_in_caption_tl
1767     {
1768       \int_gset:Nn \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1769       \int_gset:Nn \c@tabularnote \l_@@_note_in_caption_tl
1770     }
1771   }

```

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).

```

1772 \seq_gclear:N \g_@@_multicolumn_cells_seq
1773 \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).

```
1774 \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`).

```
1775 \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.

```
1776 \int_gzero:N \g_@@_col_total_int
1777 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1778 \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.

```
1779 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1780 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1781 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1782 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1783 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1784 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1785 \tl_gclear:N \g_nicematrix_code_before_tl
1786 \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]`).

```
1787 \dim_zero_new:N \l_@@_left_delim_dim
1788 \dim_zero_new:N \l_@@_right_delim_dim
1789 \bool_if:NTF \g_@@_delims_bool
1790 {
```

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

```
1791 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1792 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1793 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1794 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1795 }
1796 {
1797 \dim_gset:Nn \l_@@_left_delim_dim
1798 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1799 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1800 }
1801 }
```

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.

```
1802 \cs_new_protected:Npn \@@_pre_array:
1803 {
1804 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1805 \int_gzero_new:N \c@iRow
1806 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1807 \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).

```

1808 \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1809 { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1810 \int_compare:nNnT \l_@@_last_col_int > \c_zero_int
1811 { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1812 \bool_if:NT \g_@@_aux_found_bool
1813 {
1814   \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1815   \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1816   \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1817   \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1818 }

```

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).

```

1819 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1820 {
1821   \bool_set_true:N \l_@@_last_row_without_value_bool
1822   \bool_if:NT \g_@@_aux_found_bool
1823   { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1824 }
1825 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1826 {
1827   \bool_if:NT \g_@@_aux_found_bool
1828   { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1829 }

```

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”.

```

1830 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1831 {
1832   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1833   {
1834     \dim_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1835     { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1836     \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1837     { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1838   }
1839 }

1840 \seq_gclear:N \g_@@_cols_vlism_seq
1841 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1842 \bool_if:NT \l_@@_code_before_bool \@@_exec_code_before:

```

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

```

1843 \@@_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).

```

1844 \hbox_set:Nw \l_@@_the_array_box
1845 \skip_horizontal:N \l_@@_left_margin_dim
1846 \skip_horizontal:N \l_@@_extra_left_margin_dim
1847 \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).

```

1848 \m@th
1849 $ % $
1850 \bool_if:NTF \l_@@_light_syntax_bool
1851 { \use:c { @@-light-syntax } }
1852 { \use:c { @@-normal-syntax } }
1853 }

```

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

```

1854 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1855 {
1856   \tl_set:Nn \l_tmpa_tl { #1 }
1857   \int_compare:nNt { \char_value_catcode:n { 60 } } = { 13 }
1858   { \@@_rescan_for_spanish:N \l_tmpa_tl }
1859   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1860   \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...

```

1861 \@@_pre_array:
1862 }

```

9 The `\CodeBefore`

```

1863 \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).

```

1864 \cs_new_protected:Npn \@@_pre_code_before:
1865 {

```

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.

```

1866 \pgfsys@markposition { \@@_env: - position }
1867 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1868 \pgfpicture
1869 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1870 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1871 {
1872   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1873   \pgfcoordinate { \@@_env: - row - ##1 }
1874   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1875 }

```

Now, the recreation of the `col` nodes.

```

1876 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1877 {
1878   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:

```

```

1879     \pgfcoordinate { \@@_env: - col - ##1 }
1880     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1881 }

```

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

```

1882     \bool_if:NT \g_@@_create_cell_nodes_bool \@@_recreate_cell_nodes:
1883     \endpgfpicture

```

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

```

1884     \@@_create_diag_nodes:

```

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

```

1885     \@@_create_blocks_nodes:
1886     \IfPackageLoadedT { tikz }
1887     {
1888         \tikzset
1889         {
1890             every-picture / .style =
1891             { overlay , name~prefix = \@@_env: - }
1892         }
1893     }
1894     \cs_set_eq:NN \cellcolor \@@_cellcolor
1895     \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1896     \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1897     \cs_set_eq:NN \rowcolor \@@_rowcolor
1898     \cs_set_eq:NN \rowcolors \@@_rowcolors
1899     \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1900     \cs_set_eq:NN \arraycolor \@@_arraycolor
1901     \cs_set_eq:NN \columncolor \@@_columncolor
1902     \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1903     \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1904     \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1905     \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1906     \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1907     \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1908 }

```

```

1909 \cs_new_protected:Npn \@@_exec_code_before:
1910 {

```

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...

```

1911     \clist_map_inline:Nn \l_@@_corners_cells_clist
1912     { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1913     \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`.

```

1914     \@@_add_to_colors_seq:nn { { nocolor } } { }
1915     \bool_gset_false:N \g_@@_create_cell_nodes_bool
1916     \group_begin:

```

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

```

1917     \if_mode_math:
1918     \@@_exec_code_before_i:
1919     \else:
1920         $ % $
1921         \@@_exec_code_before_i:
1922         $ % $
1923     \fi:
1924     \group_end:
1925 }

```

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`).

```

1926 \cs_new_protected:Npn \@@_exec_code_before_i:
1927 {
1928   \int_compare:nNt { \char_value_catcode:n { 60 } } = { 13 }
1929   { \@@_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`.

```

1930   \exp_last_unbraced:No \@@_CodeBefore_keys:
1931   \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.

```

1932   \@@_actually_color:
1933   \l_@@_code_before_tl
1934   \q_stop
1935 }

```

```

1936 \keys_define:nn { nicematrix / CodeBefore }
1937 {
1938   create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1939   create-cell-nodes .default:n = true ,
1940   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1941   sub-matrix .value_required:n = true ,
1942   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1943   delimiters / color .value_required:n = true ,
1944   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1945 }

1946 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1947 {
1948   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1949   \@@_CodeBefore:w
1950 }

```

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.

```

1951 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1952 {
1953   \bool_if:NTF \g_@@_aux_found_bool
1954   {
1955     \@@_pre_code_before:
1956     \legacy_if:nF { measuring@ } { #1 }
1957   }

```

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

```

1958   {
1959     \pgfsys@markposition { \@@_env: - position }
1960     \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1961     \pgfpicture
1962     \pgf@relevantforpicturesizefalse
1963     \endpgfpicture

```

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

```

1964     \pgfpicture
1965     \pgfrememberpicturepositiononpagetrue
1966     \endpgfpicture

```


The following picture corresponds to \@@_create_blocks_nodes:.

```

1967 \pgfpicture
1968 \pgf@relevantforpicturesizefalse
1969 \pgfrememberpicturepositiononpagetrue
1970 \endpgfpicture

```

The following picture corresponds \@@_actually_color:

```

1971 \pgfpicture
1972 \pgf@relevantforpicturesizefalse
1973 \endpgfpicture
1974 }
1975 }

```

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.

```

1976 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1977 {
1978   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1979   {
1980     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1981     \pgfcoordinate { \@@_env: - row - ##1 - base }
1982     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1983     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1984     {
1985       \cs_if_exist:cT
1986       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1987       {
1988         \pgfsys@getposition
1989         { \@@_env: - ##1 - #####1 - NW }
1990         \@@_node_position:
1991         \pgfsys@getposition
1992         { \@@_env: - ##1 - #####1 - SE }
1993         \@@_node_position_i:
1994         \@@_pgf_rect_node:nnn
1995         { \@@_env: - ##1 - #####1 }
1996         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1997         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1998       }
1999     }
2000   }
2001   \@@_create_extra_nodes:
2002   \@@_create_aliases_last:
2003 }

2004 \cs_new_protected:Npn \@@_create_aliases_last:
2005 {
2006   \int_step_inline:nn \c@iRow
2007   {
2008     \pgfnodealias
2009     { \@@_env: - ##1 - last }
2010     { \@@_env: - ##1 - \int_use:N \c@jCol }
2011   }
2012   \int_step_inline:nn \c@jCol
2013   {
2014     \pgfnodealias
2015     { \@@_env: - last - ##1 }
2016     { \@@_env: - \int_use:N \c@iRow - ##1 }
2017   }
2018   \pgfnodealias
2019   { \@@_env: - last - last }
2020   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }

```

```

2021 }

2022 \cs_new_protected:Npn \@@_create_blocks_nodes:
2023 {
2024   \pgfpicture
2025   \pgf@relevantforpicturesizefalse
2026   \pgfrememberpicturepositiononpagetrue
2027   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
2028     { \@@_create_one_block_node:nnnnn ##1 }
2029   \endpgfpicture
2030 }

```

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

```

2031 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
2032 {
2033   \tl_if_empty:nF { #5 }
2034   {
2035     \@@_qpoint:n { col - #2 }
2036     \dim_set_eq:NN \l_tmpa_dim \pgf@x
2037     \@@_qpoint:n { #1 }
2038     \dim_set_eq:NN \l_tmpb_dim \pgf@y
2039     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
2040     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
2041     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
2042     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
2043     \@@_pgf_rect_node:nnnnn
2044       { \@@_env: - #5 }
2045       { \dim_use:N \l_tmpa_dim }
2046       { \dim_use:N \l_tmpb_dim }
2047       { \dim_use:N \l_@@_tmpc_dim }
2048       { \dim_use:N \l_@@_tmpd_dim }
2049   }
2050 }

2051 \cs_new_protected:Npn \@@_patch_for_revtex:
2052 {
2053   \cs_set_eq:NN \@addamp \@addamp@LaTeX
2054   \cs_set_eq:NN \@array \@array@array
2055   \cs_set_eq:NN \@tabular \@tabular@array
2056   \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
2057   \cs_set_eq:NN \array \array@array
2058   \cs_set_eq:NN \endarray \endarray@array
2059   \cs_set:Npn \endtabular { \endarray $\egroup } % $
2060   \cs_set_eq:NN \@mkpream \@mkpream@array
2061   \cs_set_eq:NN \@classx \@classx@array
2062   \cs_set_eq:NN \insert@column \insert@column@array
2063   \cs_set_eq:NN \@arraycr \@arraycr@array
2064   \cs_set_eq:NN \@xarraycr \@xarraycr@array
2065   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
2066 }

```

10 The environment `{NiceArrayWithDelims}`

```

2067 \NewDocumentEnvironment { NiceArrayWithDelims }
2068   { m m O { } m ! O { } t \CodeBefore }
2069   {

```

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).

```

2070 \bool_if:NT \c_@@_revtex_bool { \@@_patch_for_revtex: }
2071 \@@_provide_pgfsyspdfmark:
2072 \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.

```

2073 \bgroup

2074 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2075 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2076 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
2077 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

2078 \int_gzero:N \g_@@_block_box_int
2079 \dim_gzero:N \g_@@_width_last_col_dim
2080 \dim_gzero:N \g_@@_width_first_col_dim
2081 \bool_gset_false:N \g_@@_row_of_col_done_bool
2082 \str_if_empty:NT \g_@@_name_env_str
2083 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
2084 \bool_if:NTF \l_@@_tabular_bool
2085 \mode_leave_vertical:
2086 \@@_test_if_math_mode:
2087 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
2088 \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.

```

2089 \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.

```

2090 \cs_if_exist:NT \tikz@library@external@loaded
2091 {
2092 \tikzexternaldisable
2093 \cs_if_exist:NT \ifstandalone
2094 { \tikzset { external / optimize = false } }
2095 }

```

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

```

2096 \int_gincr:N \g_@@_env_int
2097 \bool_if:NF \l_@@_block_auto_columns_width_bool
2098 { \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.

```

2099 \seq_gclear:N \g_@@_blocks_seq
2100 \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`.

```

2101 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
2102 \seq_gclear:N \g_@@_pos_of_xdots_seq
2103 \tl_gclear_new:N \g_@@_code_before_tl
2104 \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.

⁸e.g. `\color[rgb]{0.5,0.5,0}`

```

2105 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2106 {
2107   \bool_gset_true:N \g_@@_aux_found_bool
2108   \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2109 }
2110 { \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.

```

2111 \tl_gclear:N \g_@@_aux_tl
2112 \tl_if_empty:NF \g_@@_code_before_tl
2113 {
2114   \bool_set_true:N \l_@@_code_before_bool
2115   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
2116 }
2117 \tl_if_empty:NF \g_@@_pre_code_before_tl
2118 { \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.

```

2119 \bool_if:NTF \g_@@_delims_bool
2120 { \keys_set:nn { nicematrix / pNiceArray } }
2121 { \keys_set:nn { nicematrix / NiceArray } }
2122 { #3 , #5 }

2123 \@@_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:.

```

2124 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
2125 }

```

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

```

2126 {
2127   \bool_if:NTF \l_@@_light_syntax_bool
2128   { \use:c { end @@-light-syntax } }
2129   { \use:c { end @@-normal-syntax } }
2130   $ % $
2131   \skip_horizontal:N \l_@@_right_margin_dim
2132   \skip_horizontal:N \l_@@_extra_right_margin_dim
2133   \hbox_set_end:
2134   \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.

```

2135 \bool_if:NT \l_@@_width_used_bool
2136 {
2137   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2138   { \@@_error_or_warning:n { width~without~X~columns } }
2139 }

```

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 .

```

2140 \fp_compare:nNnT \g_@@_total_X_weight_fp > \c_zero_fp
2141 \@@_compute_width_X:

```

It 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).

```

2142 \int_compare:nNnT \l_@@_last_row_int > { -2 }
2143 {
2144   \bool_if:NF \l_@@_last_row_without_value_bool
2145   {
2146     \int_compare:nNnF \l_@@_last_row_int = \c@iRow
2147     {
2148       \@@_error:n { Wrong~last~row }
2149       \int_set_eq:NN \l_@@_last_row_int \c@iRow
2150     }
2151   }
2152 }

```

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

```

2153 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2154 \bool_if:NTF \g_@@_last_col_found_bool
2155 { \int_gdecr:N \c@jCol }
2156 {
2157   \int_compare:nNnT \l_@@_last_col_int > { -1 }
2158   { \@@_error:n { last~col~not~used } }
2159 }

```

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

```

2160 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2161 \int_compare:nNnT \l_@@_last_row_int > { -1 }
2162 { \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. 96).

```

2163 \int_if_zero:nT \l_@@_first_col_int
2164 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2165 \bool_if:nTF { ! \g_@@_delims_bool }
2166 {
2167   \str_if_eq:eeTF c \l_@@_baseline_tl
2168   \@@_use_arraybox_with_notes_c:
2169   {
2170     \str_if_eq:eeTF b \l_@@_baseline_tl
2171     \@@_use_arraybox_with_notes_b:
2172     \@@_use_arraybox_with_notes:
2173   }
2174 }

```

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).

```

2175 {
2176   \int_if_zero:nTF \l_@@_first_row_int
2177   {
2178     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2179     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2180   }
2181   { \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

```

2182 \int_compare:nNnTF \l_@@_last_row_int > { -2 }

```

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

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).

```

2183     {
2184       \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2185       \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2186     }
2187     { \dim_zero:N \l_tmpb_dim }
2188   \hbox_set:Nn \l_tmpa_box
2189   {
2190     \m@th
2191     $ % $
2192     \@@_color:o \l_@@_delimiters_color_tl
2193     \exp_after:wN \left \g_@@_left_delim_tl
2194     \vcenter
2195     {

```

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.

```

2196       \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2197       \hbox
2198       {
2199         \bool_if:NTF \l_@@_tabular_bool
2200         { \skip_horizontal:n { - \tabcolsep } }
2201         { \skip_horizontal:n { - \arraycolsep } }
2202         \@@_use_arraybox_with_notes_c:
2203         \bool_if:NTF \l_@@_tabular_bool
2204         { \skip_horizontal:n { - \tabcolsep } }
2205         { \skip_horizontal:n { - \arraycolsep } }
2206       }

```

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

```

2207       \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2208     }
2209     \exp_after:wN \right \g_@@_right_delim_tl
2210     $ % $
2211   }

```

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.

```

2212     \bool_if:NTF \l_@@_delimiters_max_width_bool
2213     { \@@_put_box_in_flow_bis:nn \g_@@_left_delim_tl \g_@@_right_delim_tl }
2214     \@@_put_box_in_flow:
2215   }

```

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. 97).

```

2216     \bool_if:NT \g_@@_last_col_found_bool
2217     { \skip_horizontal:N \g_@@_width_last_col_dim }
2218     \bool_if:NT \l_@@_preamble_bool
2219     {
2220       \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2221       { \@@_err_columns_not_used: }
2222     }
2223     \@@_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.

```

2224     \egroup

```

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

```

2225     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2226     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2227     \iow_now:Ne \@mainaux
2228     {
2229       \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2230       \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }

```

```

2231         { \exp_not:o \g_@@_aux_tl }
2232     }
2233     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2234     \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2235 }

```

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

```

2236 \cs_new_protected:Npn \@@_err_columns_not_used:
2237 {
2238     \@@_warning:n { columns~not~used }
2239     \cs_gset:Npn \@@_err_columns_not_used: { }
2240 }

```

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 .

```

2241 \cs_new_protected:Npn \@@_compute_width_X:
2242 {
2243     \tl_gput_right:Ne \g_@@_aux_tl
2244     {
2245         \bool_set_true:N \l_@@_X_columns_aux_bool
2246         \dim_set:Nn \l_@@_X_columns_dim
2247     }

```

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`

```

2248     \bool_lazy_and:nnTF \g_@@_V_of_X_bool \l_@@_X_columns_aux_bool
2249     { \dim_use:N \l_@@_X_columns_dim }
2250     {
2251         \dim_compare:nNnTF
2252         {
2253             \dim_abs:n
2254             { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2255         }
2256         <
2257         { 0.001 pt }
2258         { \dim_use:N \l_@@_X_columns_dim }
2259         {
2260             \dim_eval:n
2261             {
2262                 \l_@@_X_columns_dim
2263                 +
2264                 \fp_to_dim:n
2265                 {
2266                     (
2267                         \dim_eval:n
2268                         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2269                     )
2270                     / \fp_use:N \g_@@_total_X_weight_fp
2271                 }
2272             }
2273         }
2274     }
2275 }
2276 }
2277 }

```

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`.

```

2278 \cs_new_protected:Npn \@@_transform_preamble:
2279 {
2280   \@@_transform_preamble_i:
2281   \@@_transform_preamble_ii:
2282 }
2283 \cs_new_protected:Npn \@@_transform_preamble_i:
2284 {
2285   \int_gzero:N \c@jCol

```

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

```

2286   \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.

```

2287   \bool_gset_false:N \g_tmpb_bool

```

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

```

2288   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2289   \int_zero:N \l_tmpa_int
2290   \tl_gclear:N \g_@@_array_preamble_tl
2291   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2292   {
2293     \tl_gset:Nn \g_@@_array_preamble_tl
2294     { ! { \skip_horizontal:N \arrayrulewidth } }
2295   }
2296   {
2297     \clist_if_in:NnT \l_@@_vlines_clist 1
2298     {
2299       \tl_gset:Nn \g_@@_array_preamble_tl
2300       { ! { \skip_horizontal:N \arrayrulewidth } }
2301     }
2302   }

```

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

```

2303   \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2304   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2305   \@@_replace_columncolor:
2306 }

2307 \cs_new_protected:Npn \@@_transform_preamble_ii:
2308 {

```

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

```

2309   \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2310   {
2311     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2312     { \bool_gset_true:N \g_@@_delims_bool }
2313   }
2314   { \bool_gset_true:N \g_@@_delims_bool }

```


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

```
2315 \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).

```
2316 \int_if_zero:nTF \l_@@_first_col_int
2317 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2318 {
2319   \bool_if:NF \g_@@_delims_bool
2320   {
2321     \bool_if:NF \l_@@_tabular_bool
2322     {
2323       \clist_if_empty:NT \l_@@_vlines_clist
2324       {
2325         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2326         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2327       }
2328     }
2329   }
2330 }
2331 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2332 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2333 {
2334   \bool_if:NF \g_@@_delims_bool
2335   {
2336     \bool_if:NF \l_@@_tabular_bool
2337     {
2338       \clist_if_empty:NT \l_@@_vlines_clist
2339       {
2340         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2341         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2342       }
2343     }
2344   }
2345 }
```

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.

```
2346 \tag_if_active:F
2347 {
```

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

```
2348 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2349 {
2350   \tl_gput_right:Nn \g_@@_array_preamble_tl
2351   { > { \@@_err_too_many_cols: } 1 }
2352 }
2353 }
2354 }
```

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.

```
2355 \cs_new_protected:Npn \@@_rec_preamble:n #1
2356 {
```

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

```
2357 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2358 { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2359 {
```

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

```
2360 \cs_if_exist:cTF { NC @ find @ #1 }
2361 {
2362 \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2363 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2364 }
2365 {
2366 \str_if_eq:nnTF { #1 } { S }
2367 { \@@_fatal:n { unknown~column~type~S } }
2368 { \@@_fatal:nn { unknown~column~type } { #1 } }
2369 }
2370 }
2371 }
```

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

```
2372 \cs_new_protected:Npn \@@_c: #1
2373 {
2374 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2375 \tl_gclear:N \g_@@_pre_cell_tl
2376 \tl_gput_right:Nn \g_@@_array_preamble_tl
2377 { > \@@_cell_begin: c < \@@_cell_end: }
```

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

```
2378 \int_gincr:N \c@jCol
2379 \@@_rec_preamble_after_col:n
2380 }

2381 \cs_new_protected:Npn \@@_l: #1
2382 {
2383 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2384 \tl_gclear:N \g_@@_pre_cell_tl
2385 \tl_gput_right:Nn \g_@@_array_preamble_tl
2386 {
2387 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2388 l
2389 < \@@_cell_end:
2390 }
2391 \int_gincr:N \c@jCol
2392 \@@_rec_preamble_after_col:n
2393 }

2394 \cs_new_protected:Npn \@@_r: #1
2395 {
2396 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2397 \tl_gclear:N \g_@@_pre_cell_tl
2398 \tl_gput_right:Nn \g_@@_array_preamble_tl
2399 {
2400 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2401 r
2402 < \@@_cell_end:
2403 }
2404 \int_gincr:N \c@jCol
2405 \@@_rec_preamble_after_col:n
2406 }
```

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`.

For ! and @

```

2407 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2408 {
2409     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2410     \@@_rec_preamble:n
2411 }
2412 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2413 \cs_new_protected:cpn { @@ _ | : } #1
2414 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2415     \int_incr:N \l_tmpa_int
2416     \@@_make_preamble_i_i:n
2417 }
2418 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2419 {

```

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

```

2420     \str_if_eq:nnTF { #1 } { | }
2421     { \use:c { @@ _ | : } | }
2422     { \@@_make_preamble_i_ii:nn { } #1 }
2423 }

```

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

```

2424 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2425 {
2426     \str_if_eq:nnTF { #2 } { [ ] }
2427     { \@@_make_preamble_i_ii:nw { #1 } [ ] }
2428     { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2429 }
2430 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2431 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2432 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2433 {
2434     \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2435     \tl_gput_right:Ne \g_@@_array_preamble_tl
2436     {

```

Here, the command \dim_use:N is mandatory.

```

2437         \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2438     }
2439     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2440     {
2441         \@@_vline:n
2442         {
2443             position = \int_eval:n { \c@jCol + 1 } ,
2444             multiplicity = \int_use:N \l_tmpa_int ,
2445             total-width = \dim_use:N \l_@@_rule_width_dim ,
2446             #2
2447         }

```

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.

```

2448     }
2449     \int_zero:N \l_tmpa_int
2450     \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2451     \@@_rec_preamble:n #1
2452 }

```

```

2453 \cs_new_protected:cpn { @@ _ > : } #1 #2
2454 {
2455   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2456   \@@_rec_preamble:n
2457 }
2458 \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.

```

2459 \keys_define:nn { nicematrix / p-column }
2460 {
2461   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2462   r .value_forbidden:n = true ,
2463   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2464   c .value_forbidden:n = true ,
2465   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2466   l .value_forbidden:n = true ,
2467   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2468   S .value_forbidden:n = true ,
2469   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2470   p .value_forbidden:n = true ,
2471   t .meta:n = p ,
2472   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2473   m .value_forbidden:n = true ,
2474   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2475   b .value_forbidden:n = true
2476 }

```

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

```

2477 \cs_new_protected:Npn \@@_p: #1
2478 {
2479   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2480   \@@_make_preamble_ii_i:n
2481 }
2482 \cs_new_eq:NN \@@_b: \@@_p:
2483 \cs_new_eq:NN \@@_m: \@@_p:
2484 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2485 {
2486   \str_if_eq:nnTF { #1 } { [ ]
2487     { \@@_make_preamble_ii_ii:w [ ]
2488       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2489     }
2490 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2491 { \@@_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.

```

2492 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2493 {

```

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).

```

2494   \str_set:Nn \l_@@_hpos_col_str { j }
2495   \@@_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`.

```

2496   \setlength { \l_tmpa_dim } { #2 }

```

```

2497 \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2498 }
2499 \cs_new_protected:Npn \@@_keys_p_column:n #1
2500 { \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.

```

2501 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2502 {

```

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

```

2503 \use:e
2504 {
2505 \@@_make_preamble_ii_vi:nnnnnnnn
2506 { \str_if_eq:eeTF p \l_@@_vpos_col_str { t } { b } }
2507 { #1 }
2508 {
2509 \cs_set_eq:NN \rotate \@@_rotate_p_col:

```

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.

```

2510 \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2511 { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2512 {

```

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

```

2513 \def \exp_not:N \l_@@_hpos_cell_tl
2514 { \str_lowercase:f { \l_@@_hpos_col_str } }
2515 }
2516 \IfPackageLoadedTF { ragged2e }
2517 {
2518 \str_case:on \l_@@_hpos_col_str
2519 {

```

The following `\exp_not:N` are mandatory.

```

2520 c { \exp_not:N \Centering }
2521 l { \exp_not:N \RaggedRight }
2522 r { \exp_not:N \RaggedLeft }
2523 }
2524 }
2525 {
2526 \str_case:on \l_@@_hpos_col_str
2527 {
2528 c { \exp_not:N \centering }
2529 l { \exp_not:N \raggedright }
2530 r { \exp_not:N \raggedleft }
2531 }
2532 }
2533 #3
2534 }
2535 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2536 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2537 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2538 { #2 }
2539 {
2540 \str_case:onF \l_@@_hpos_col_str
2541 {
2542 { j } { c }
2543 { si } { c }
2544 }

```

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

```
2545         { \str_lowercase:f \l_@@_hpos_col_str }
2546     }
2547 }
```

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

```
2548     \int_gincr:N \c@jCol
2549     \@@_rec_preamble_after_col:n
2550 }
```

#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*.

```
2551 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2552 {
2553     \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2554     {
2555         \tl_gput_right:Nn \g_@@_array_preamble_tl
2556         { > \@@_test_if_empty_for_S: }
2557     }
2558     {
2559         \str_if_eq:eeTF { #7 } { varwidth }
2560         {
2561             \tl_gput_right:Nn \g_@@_array_preamble_tl
2562             { > \@@_test_if_empty_varwidth: }
2563         }
2564         { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2565     }
2566     \tl_gput_right:Nn \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2567     \tl_gclear:N \g_@@_pre_cell_tl
2568     \tl_gput_right:Nn \g_@@_array_preamble_tl
2569     {
2570         > {
```

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

```
2571         \dim_set:Nn \l_@@_col_width_dim { #2 }
2572         \@@_cell_begin:
```

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

```
2573         \use:c { #7 } [ #1 ] { #2 }
```

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

```
2574     \everypar
2575     {
2576         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2577         \everypar { }
2578     }
```

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

```
2579     #3
```

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

```

2580         \g_@@_row_style_tl
2581         \arraybackslash
2582         #5
2583     }
2584     #8
2585     < {
2586         #6

```

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

```

2587         \@finalstrut \@arstrutbox
2588         \use:c { end #7 }

```

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

```

2589         #4
2590         \@@_cell_end:
2591     }
2592 }
2593 }

```

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

```

2594 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2595 {

```

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 `&`).

```

2596 \group_align_safe_begin:
2597 \peek_meaning:NTF &
2598 \@@_the_cell_is_empty:
2599 {
2600     \peek_meaning:NTF \\\
2601     \@@_the_cell_is_empty:
2602     {
2603         \peek_meaning:NTF \crcr
2604         \@@_the_cell_is_empty:
2605         \group_align_safe_end:
2606     }
2607 }
2608 }

```

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

```

2609 \cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
2610 {
2611     \group_align_safe_begin:
2612     \peek_meaning:NTF &
2613     \@@_the_cell_is_empty_varwidth:
2614     {
2615         \peek_meaning:NTF \\\
2616         \@@_the_cell_is_empty_varwidth:
2617         {
2618             \peek_meaning:NTF \crcr
2619             \@@_the_cell_is_empty_varwidth:
2620             \group_align_safe_end:
2621         }
2622     }
2623 }
2624 \cs_new_protected:Npn \@@_the_cell_is_empty:
2625 {
2626     \group_align_safe_end:
2627     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2628     {

```

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

```
2629 \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
```

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

```
2630 \skip_horizontal:N \l_@@_col_width_dim
```

```
2631 }
```

```
2632 }
```

```
2633 \cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
```

```
2634 {
```

```
2635 \group_align_safe_end:
```

```
2636 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
```

```
2637 { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
```

```
2638 }
```

```
2639 \cs_new_protected:Npn \@@_test_if_empty_for_S:
```

```
2640 {
```

```
2641 \peek_meaning:NT \__siunitx_table_skip:n
```

```
2642 { \bool_gset_true:N \g_@@_empty_cell_bool }
```

```
2643 }
```

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.

```
2644 \cs_new_protected:Npn \@@_center_cell_box:
```

```
2645 {
```

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

```
2646 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
```

```
2647 {
```

```
2648 \dim_compare:nNnT
```

```
2649 { \box_ht:N \l_@@_cell_box }
```

```
2650 >
```

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).

```
2651 { \box_ht:N \strutbox }
```

```
2652 {
```

```
2653 \hbox_set:Nn \l_@@_cell_box
```

```
2654 {
```

```
2655 \box_move_down:nn
```

```
2656 {
```

```
2657 ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
```

```
2658 + \baselineskip ) / 2
```

```
2659 }
```

```
2660 { \box_use:N \l_@@_cell_box }
```

```
2661 }
```

```
2662 }
```

```
2663 }
```

```
2664 }
```

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

```
2665 \cs_new_protected:Npn \@@_V: #1 #2
```

```
2666 {
```

```
2667 \str_if_eq:nnTF { #2 } { [ ] }
```

```
2668 { \@@_make_preamble_V_i:w [ ] }
```

```
2669 { \@@_make_preamble_V_i:w [ ] { #2 } }
```

```
2670 }
```



```

2671 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2672 { \@@_make_preamble_V_ii:nn { #1 } }
2673 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2674 {
2675   \str_set:Nn \l_@@_vpos_col_str { p }
2676   \str_set:Nn \l_@@_hpos_col_str { j }
2677   \@@_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`.

```

2678   \setlength { \l_tmpa_dim } { #2 }
2679   \IfPackageLoadedTF { varwidth }
2680   { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2681   {
2682     \@@_error_or_warning:n { varwidth-not-loaded }
2683     \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2684   }
2685 }
2686 % \end{macrocod}
2687 %
2688 % \medskip
2689 % For |w| and |W|
2690 % \begin{macrocode}
2691 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2692 \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. It is provided by curryfication.

```

2693 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3
2694 {
2695   \tl_if_in:nnTF { clr } { #3 }
2696   { \@@_make_preamble_w_i:nnnn { #1 } { #2 } { #3 } }
2697   {
2698     \@@_error:nn { Invalid-argument-for-w } { #3 }
2699     \@@_make_preamble_w_i:nnnn { #1 } { #2 } { c }
2700   }
2701 }
2702 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2 #3 #4
2703 {
2704   \str_if_eq:nnTF { #3 } { s }
2705   { \@@_make_preamble_w_ii:nnnn { #1 } { #4 } }
2706   { \@@_make_preamble_w_iii:nnnn { #1 } { #2 } { #3 } { #4 } }
2707 }

```

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.

```

2708 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2
2709 {
2710   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2711   \tl_gclear:N \g_@@_pre_cell_tl
2712   \tl_gput_right:Nn \g_@@_array_preamble_tl
2713   {
2714     > {

```

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`.

```

2715         \setlength { \l_@@_col_width_dim } { #2 }
2716         \@@_cell_begin:
2717         \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2718     }
2719     c
2720     < {
2721         \@@_cell_end_for_w_s:
2722         #1
2723         \@@_adjust_size_box:
2724         \box_use_drop:N \l_@@_cell_box
2725     }
2726 }
2727 \int_gincr:N \c@jCol
2728 \@@_rec_preamble_after_col:n
2729 }

```

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

```

2730 \cs_new_protected:Npn \@@_make_preamble_w_iii:nnnn #1 #2 #3 #4
2731 {
2732     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2733     \tl_gclear:N \g_@@_pre_cell_tl
2734     \tl_gput_right:Nn \g_@@_array_preamble_tl
2735     {
2736         > {

```

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`.

```

2737         \setlength { \l_@@_col_width_dim } { #4 }
2738         \hbox_set:Nw \l_@@_cell_box
2739         \@@_cell_begin:
2740         \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2741     }
2742     c
2743     < {
2744         \@@_cell_end:
2745         \hbox_set_end:
2746         #1
2747         \@@_adjust_size_box:
2748         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2749     }
2750 }

```

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

```

2751     \int_gincr:N \c@jCol
2752     \@@_rec_preamble_after_col:n
2753 }

```

```

2754 \cs_new_protected:Npn \@@_special_W:
2755 {
2756     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2757     { \@@_warning:n { W-warning } }
2758 }

```

For S (of siunitx).

```

2759 \cs_new_protected:Npn \@@_S: #1 #2
2760 {
2761     \str_if_eq:nnTF { #2 } { [ ] }
2762     { \@@_make_preamble_S:w [ ] }
2763     { \@@_make_preamble_S:w [ ] { #2 } }
2764 }

```

```

2765 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2766 { \@@_make_preamble_S_i:n { #1 } }
2767 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2768 {
2769   \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2770   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2771   \tl_gclear:N \g_@@_pre_cell_tl
2772   \tl_gput_right:Nn \g_@@_array_preamble_tl
2773   {
2774     > {

```

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).

```

2775       \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2776       \keys_set:nn { siunitx } { #1 }
2777       \@@_cell_begin:
2778       \siunitx_cell_begin:w
2779     }
2780     c
2781     <
2782     {
2783       \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`).

```

2784       \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2785       {
2786         \bool_if:NTF \l__siunitx_table_text_bool
2787         \bool_set_true:N
2788         \bool_set_false:N
2789         \l__siunitx_table_text_bool
2790       }
2791       \@@_cell_end:
2792     }
2793   }

```

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

```

2794       \int_gincr:N \c@jCol
2795       \@@_rec_preamble_after_col:n
2796     }

```

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

```

2797 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2798 {
2799   \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.

```

2800   \int_if_zero:nTF \c@jCol
2801   {
2802     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2803     {

```

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

```

2804       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2805       \tl_gset_eq:Nn \g_@@_right_delim_tl \c_@@_dot_tl
2806       \@@_rec_preamble:n #2
2807     }
2808   {
2809     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }

```

```

2810         \@@_make_preamble_iv:nn { #1 } { #2 }
2811     }
2812 }
2813 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2814 }
2815 \cs_set_eq:cc { @@ _ \token_to_str:N [ : ] { @@ _ \token_to_str:N ( : }
2816 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2817 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2818 {
2819     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2820     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2821     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2822     {
2823         \@@_error:nn { delimiter~after~opening } { #2 }
2824         \@@_rec_preamble:n
2825     }
2826     { \@@_rec_preamble:n #2 }
2827 }

```

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

```

2828 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2829 { \use:c { @@ _ \token_to_str:N ( : } }

```

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

```

2830 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2831 {
2832     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2833     \tl_if_in:nnTF { ) ] \} } { #2 }
2834     { \@@_make_preamble_v:nnn #1 #2 }
2835     {
2836         \str_if_eq:nnTF \s_stop { #2 }
2837         {
2838             \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2839             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2840             {
2841                 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2842                 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2843                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2844                 \@@_rec_preamble:n #2
2845             }
2846         }
2847         {
2848             \tl_if_in:nnT { ( [ \{ \left } { #2 }
2849             { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2850             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2851             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2852             \@@_rec_preamble:n #2
2853         }
2854     }
2855 }
2856 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2857 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2858 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2859 {
2860     \str_if_eq:nnTF \s_stop { #3 }
2861     {
2862         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2863         {
2864             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }

```

```

2865 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2866 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2867 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2868 }
2869 {
2870 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2871 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2872 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2873 \@@_error:nn { double~closing~delimiter } { #2 }
2874 }
2875 }
2876 {
2877 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2878 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2879 \@@_error:nn { double~closing~delimiter } { #2 }
2880 \@@_rec_preamble:n #3
2881 }
2882 }

2883 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2884 { \use:c { @@ _ \token_to_str:N ) : } }

```

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

```

2885 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2886 {
2887 \str_if_eq:nnTF { #1 } { < }
2888 { \@@_rec_preamble_after_col_i:n }
2889 {
2890 \str_if_eq:nnTF { #1 } { @ }
2891 { \@@_rec_preamble_after_col_ii:n }
2892 {
2893 \str_if_eq:eeTF \l_@@_vlines_clist { all }
2894 {
2895 \tl_gput_right:Nn \g_@@_array_preamble_tl
2896 { ! { \skip_horizontal:N \arrayrulewidth } }
2897 }
2898 {
2899 \clist_if_in:NiT \l_@@_vlines_clist
2900 { \int_eval:n { \c@jCol + 1 } }
2901 {
2902 \tl_gput_right:Nn \g_@@_array_preamble_tl
2903 { ! { \skip_horizontal:N \arrayrulewidth } }
2904 }
2905 }
2906 \@@_rec_preamble:n { #1 }
2907 }
2908 }
2909 }

2910 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2911 {
2912 \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2913 \@@_rec_preamble_after_col:n
2914 }

```

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

```

2915 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2916 {
2917 \str_if_eq:eeTF \l_@@_vlines_clist { all }
2918 {

```

```

2919 \tl_gput_right:Nn \g_@@_array_preamble_tl
2920 { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2921 }
2922 {
2923 \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2924 {
2925 \tl_gput_right:Nn \g_@@_array_preamble_tl
2926 { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2927 }
2928 { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2929 }
2930 \@@_rec_preamble:n
2931 }

2932 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2933 {
2934 \tl_clear:N \l_tmpa_tl
2935 \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2936 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2937 }

```

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

```

2938 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2939 { \@@_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.

```

2940 \cs_new_protected:Npn \@@_X: #1 #2
2941 {
2942 \str_if_eq:nnTF { #2 } { [ ]
2943 { \@@_make_preamble_X:w [ ] }
2944 { \@@_make_preamble_X:w [ ] #2 }
2945 }
2946 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2947 { \@@_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`.

```

2948 \keys_define:nn { nicematrix / X-column }
2949 {
2950 V .code:n =
2951 \IfPackageLoadedTF { varwidth }
2952 {
2953 \bool_set_true:N \l_@@_V_of_X_bool
2954 \bool_gset_true:N \g_@@_V_of_X_bool
2955 }
2956 { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
2957 unknown .code:n =
2958 \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2959 { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2960 { \@@_error_or_warning:n { invalid~weight } }
2961 }

```

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

```

2962 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2963 {

```

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`).

```
2964 \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`).

```
2965 \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`.

```
2966 \fp_set:Nn \l_tmpa_fp { 1.0 }
```

```
2967 \@@_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.

```
2968 \bool_set_false:N \l_@@_V_of_X_bool
```

```
2969 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
```

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

```
2970 \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).

```
2971 \bool_if:NTF \l_@@_X_columns_aux_bool
```

```
2972 {
```

```
2973 \@@_make_preamble_ii_iv:nnn
```

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

```
2974 { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
```

```
2975 { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
```

```
2976 { \@@_no_update_width: }
```

```
2977 }
```

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.

```
2978 {
```

```
2979 \tl_gput_right:Nn \g_@@_array_preamble_tl
```

```
2980 {
```

```
2981 > {
```

```
2982 \@@_cell_begin:
```

```
2983 \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`.

```
2984 \NotEmpty
```

The following code will nullify the box of the cell.

```
2985 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
```

```
2986 { \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`.

```
2987 \begin { minipage } { 5 cm } \arraybackslash
```

```
2988 }
```

```
2989 c
```

```
2990 < {
```

```
2991 \end { minipage }
```

```
2992 \@@_cell_end:
```

```
2993 }
```

```
2994 }
```

```
2995 \int_gincr:N \c@jCol
```

```

2996         \@@_rec_preamble_after_col:n
2997     }
2998 }

2999 \cs_new_protected:Npn \@@_no_update_width:
3000 {
3001     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
3002     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
3003 }

```

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

```

3004 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
3005 {
3006     \seq_gput_right:Ne \g_@@_cols_vlism_seq
3007     { \int_eval:n { \c@jCol + 1 } }
3008     \tl_gput_right:Ne \g_@@_array_preamble_tl
3009     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
3010     \@@_rec_preamble:n
3011 }

```

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.

```

3012 \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, for example, forgotten the preamble of its environment).

```

3013 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
3014 { \@@_fatal:n { Preamble~forgotten } }
3015 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
3016 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
3017 { @@ _ \token_to_str:N \hline : }
3018 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
3019 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
3020 { @@ _ \token_to_str:N \hline : }
3021 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
3022 { @@ _ \token_to_str:N \hline : }
3023 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
3024 { @@ _ \token_to_str:N \hline : }
3025 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
3026 { @@ _ \token_to_str:N \hline : }
3027 \cs_new_protected:cpn { @@ _ \token_to_str:N \linewidth : }
3028 { \@@_fatal:n { NiceTabularX~probably~required } }
3029 \cs_set_eq:cc { @@ _ \token_to_str:N \textwidth : }
3030 { @@ _ \token_to_str:N \linewidth : }

```

12 The redefinition of `\multicolumn`

The following command must *not* be protected since it begins with `\multispan` (a TeX primitive).

```

3031 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
3032 {

```

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

```

3033     \multispan { #1 }
3034     \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
3035     \begingroup
3036     \tbl_update_multicolumn_cell_data:n { #1 }

```


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

```
3037 \tl_gclear:N \g_@@_preamble_tl
3038 \@@_make_m_preamble:n #2 \q_stop
```

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

```
3039 \def \@addamp
3040 {
3041   \legacy_if:nTF { @firstamp }
3042   { \legacy_if_set_false:n { @firstamp } }
3043   { \@preamerr 5 }
3044 }
3045 \exp_args:No \@mkpream \g_@@_preamble_tl
3046 \@addtopreamble \@empty
3047 \endgroup
3048 \UseTaggingSocket { tbl / colspan } { #1 }
```

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

```
3049 \int_compare:nNtT { #1 } > 1
3050 {
3051   \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
3052   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
3053   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
3054   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
3055   {
3056     {
3057       \int_if_zero:nTF \c@jCol
3058       { \int_eval:n { \c@iRow + 1 } }
3059       { \int_use:N \c@iRow }
3060     }
3061     { \int_eval:n { \c@jCol + 1 } }
3062     {
3063       \int_if_zero:nTF \c@jCol
3064       { \int_eval:n { \c@iRow + 1 } }
3065       { \int_use:N \c@iRow }
3066     }
3067     { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block.

```
3068   { }
3069 }
3070 }
```

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

```
3071 \RenewDocumentCommand { \cellcolor } { 0 { } m }
3072 {
3073   \tl_gput_right:Ne \g_@@_pre_code_before_tl
3074   {
3075     \@_rectanglecolor [ ##1 ]
3076     { \exp_not:n { ##2 } }
3077     { \int_use:N \c@iRow - \int_use:N \c@jCol }
3078     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
3079   }
3080   \ignorespaces
3081 }
```

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

```
3082 \def \@sharp { #3 }
3083 \@arstrut
3084 \@preamble
3085 \null
```

We add some lines.

```

3086 \int_gadd:Nn \c@jCol { #1 - 1 }
3087 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
3088 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3089 \ignorespaces
3090 }

```

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`.

```

3091 \cs_new_protected:Npn \@@_make_m_preamble:n #1
3092 {
3093   \str_case:nnF { #1 }
3094   {
3095     c { \@@_make_m_preamble_i:n #1 }
3096     l { \@@_make_m_preamble_i:n #1 }
3097     r { \@@_make_m_preamble_i:n #1 }
3098     > { \@@_make_m_preamble_ii:nn #1 }
3099     ! { \@@_make_m_preamble_ii:nn #1 }
3100     @ { \@@_make_m_preamble_ii:nn #1 }
3101     | { \@@_make_m_preamble_iii:n #1 }
3102     p { \@@_make_m_preamble_iv:nnn t #1 }
3103     m { \@@_make_m_preamble_iv:nnn c #1 }
3104     b { \@@_make_m_preamble_iv:nnn b #1 }
3105     w { \@@_make_m_preamble_v:nnnn { } #1 }
3106     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
3107     \q_stop { }
3108   }
3109   {
3110     \cs_if_exist:cTF { NC @ find @ #1 }
3111     {
3112       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
3113       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
3114     }
3115     {
3116       \str_if_eq:nnTF { #1 } { S }
3117       { \@@_fatal:n { unknown~column~type~S~multicolumn } }
3118       { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
3119     }
3120   }
3121 }

```

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

```

3122 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
3123 {
3124   \tl_gput_right:Nn \g_@@_preamble_tl
3125   {
3126     > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
3127     #1
3128     < \@@_cell_end:
3129   }

```

We test for the presence of a `<`.

```

3130 \@@_make_m_preamble_x:n
3131 }

```

For `>`, `!` and `@`

```

3132 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
3133 {
3134   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
3135   \@@_make_m_preamble:n
3136 }

```

For l

```

3137 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
3138 {
3139   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
3140   \@@_make_m_preamble:n
3141 }

```

For p, m and b

```

3142 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
3143 {
3144   \tl_gput_right:Nn \g_@@_preamble_tl
3145   {
3146     > {
3147       \@@_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`.

```

3148       \setlength { \l_tmpa_dim } { #3 }
3149       \begin { minipage } [ #1 ] { \l_tmpa_dim }
3150       \mode_leave_vertical:
3151       \arraybackslash
3152       \vrule height \box_ht:N \@@arstrutbox depth \c_zero_dim width \c_zero_dim
3153     }
3154     c
3155     < {
3156       \vrule height \c_zero_dim depth \box_dp:N \@@arstrutbox width \c_zero_dim
3157       \end { minipage }
3158       \@@_cell_end:
3159     }
3160   }

```

We test for the presence of a `<`.

```

3161   \@@_make_m_preamble_x:n
3162 }

```

For w and W

```

3163 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3164 {
3165   \tl_gput_right:Nn \g_@@_preamble_tl
3166   {
3167     > {
3168       \dim_set:Nn \l_@@_col_width_dim { #4 }
3169       \hbox_set:Nw \l_@@_cell_box
3170       \@@_cell_begin:
3171       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3172     }
3173     c
3174     < {
3175       \@@_cell_end:
3176       \hbox_set_end:
3177       \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3178       #1
3179       \@@_adjust_size_box:
3180       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3181     }
3182   }

```

We test for the presence of a `<`.

```

3183   \@@_make_m_preamble_x:n
3184 }

```

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

```

3185 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3186 {
3187   \str_if_eq:nnTF { #1 } { < }
3188     \@@_make_m_preamble_ix:n
3189     { \@@_make_m_preamble:n { #1 } }
3190 }
3191 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3192 {
3193   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3194   \@@_make_m_preamble_x:n
3195 }

```

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`).

```

3196 \cs_new_protected:Npn \@@_put_box_in_flow:
3197 {
3198   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3199   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3200   \str_if_eq:eeTF \l_@@_baseline_tl { c }
3201     { \box_use_drop:N \l_tmpa_box }
3202   \@@_put_box_in_flow_i:
3203 }

```

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

```

3204 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3205 {
3206   \pgfpicture
3207     \@@_qpoint:n { row - 1 }
3208     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3209     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3210     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3211     \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).

```

3212   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3213   {
3214     \int_set:Nn \l_tmpa_int
3215       { \str_range:Nnn \l_@@_baseline_tl { 6 } { -1 } }
3216     \bool_lazy_or:nnT
3217       { \int_compare_p:nNn \l_tmpa_int < { 1 } }
3218       { \int_compare_p:nNn \l_tmpa_int > { \c@iRow + 1 } }
3219     {
3220       \@@_error:n { bad-value-for-baseline-line }
3221       \int_set:Nn \l_tmpa_int 1
3222     }
3223     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3224   }
3225   {
3226     \str_if_eq:eeTF t \l_@@_baseline_tl
3227       { \int_set:Nn \l_tmpa_int 1 }
3228       {
3229         \str_if_eq:eeTF b \l_@@_baseline_tl
3230           { \int_set_eq:NN \l_tmpa_int \c@iRow }
3231           { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3232       }
3233     \bool_lazy_or:nnT

```

```

3234         { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3235         { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3236         {
3237             \@@_error:n { bad-value~for~baseline }
3238             \int_set:Nn \l_tmpa_int 1
3239         }
3240         \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3241         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3242     }
3243     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3244     \endpgfpicture
3245     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3246     \box_use_drop:N \l_tmpa_box
3247 }

```

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).

```

3248 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3249 {

```

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.

```

3250     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3251     {
3252         \int_compare:nNnT \c@jCol > 1
3253         {
3254             \box_set_wd:Nn \l_@@_the_array_box
3255             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3256         }
3257     }

```

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

```

3258     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3259     \bool_if:NT \l_@@_caption_above_bool
3260     {
3261         \tl_if_empty:NF \l_@@_caption_tl
3262         {
3263             \bool_set_false:N \g_@@_caption_finished_bool
3264             \int_gzero:N \c@tablarnote
3265             \@@_insert_caption:

```

If there is one or several commands `\tablarnote` 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 `\tablarnote` has been used without its optional argument (between square brackets).

```

3266         \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3267         {
3268             \tl_gput_right:Ne \g_@@_aux_tl
3269             {
3270                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3271                 { \int_use:N \g_@@_notes_caption_int }
3272             }
3273             \int_gzero:N \g_@@_notes_caption_int
3274         }
3275     }
3276 }

```

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

```

3277 \hbox
3278 {
3279 \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.

```

3280 \@@_create_extra_nodes:
3281 \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3282 }

```

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).

```

3283 \bool_lazy_any:nT
3284 {
3285 { ! \seq_if_empty_p:N \g_@@_notes_seq }
3286 { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3287 { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3288 }
3289 {
3290 \bool_if:NTF \l_@@_notes_no_print_bool
3291 { \cs_get_eq:NN \NiceTabularNotes \@@_tabular_notes: }
3292 \@@_tabular_notes:
3293 }
3294 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3295 \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3296 \end { minipage }
3297 }

```

```

3298 \cs_new_protected:Npn \@@_insert_caption:
3299 {
3300 \tl_if_empty:NF \l_@@_caption_tl
3301 {
3302 \cs_if_exist:NTF \@capytype
3303 { \@@_insert_caption_i: }
3304 { \@@_error:n { caption~outside~float } }
3305 }
3306 }

```

```

3307 \cs_new_protected:Npn \@@_insert_caption_i:
3308 {
3309 \group_begin:

```

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

```

3310 \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.

```

3311 \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
3312 \tl_if_empty:NTF \l_@@_short_caption_tl
3313 \caption
3314 { \caption [ \l_@@_short_caption_tl ] }
3315 { \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.

```

3316   \bool_if:NF \g_@@_caption_finished_bool
3317   {
3318     \bool_gset_true:N \g_@@_caption_finished_bool
3319     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3320     \int_gzero:N \c@tabularnote
3321   }
3322   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3323   \group_end:
3324 }

3325 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3326 {
3327   \@@_error_or_warning:n { tabularnote-below-the-tabular }
3328   \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3329 }

3330 \cs_set_protected:Npn \@@_tabular_notes_error:
3331 { \@@_error:n { Bad-use-of-NiceTabularNotes } }
3332 \cs_set_eq:NN \NiceTabularNotes \@@_tabular_notes_error:

3333 \cs_set_protected:Npn \@@_tabular_notes:
3334 {
3335   \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes_error:
3336   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3337   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3338   \skip_vertical:N 0.65ex

```

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

```

3339   \group_begin:
3340   \l_@@_notes_code_before_tl
3341   \tl_if_empty:NF \g_@@_tabularnote_tl
3342   {
3343     \g_@@_tabularnote_tl \par
3344     \tl_gclear:N \g_@@_tabularnote_tl
3345   }

```

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.

```

3346   \int_compare:nNnT \c@tabularnote > \c_zero_int
3347   {
3348     \bool_if:NTF \l_@@_notes_para_bool
3349     {
3350       \begin { tabularnotes* }
3351         \seq_map_inline:Nn \g_@@_notes_seq
3352         { \@@_one_tabularnote:nm ##1 }
3353         \strut
3354       \end { tabularnotes* }

```

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

```

3355       \par
3356     }
3357   {
3358     \tabularnotes
3359     \seq_map_inline:Nn \g_@@_notes_seq
3360     { \@@_one_tabularnote:nm ##1 }
3361     \strut
3362     \endtabularnotes
3363   }
3364 }

```

```

3365 \unskip
3366 \group_end:
3367 \bool_if:NT \l_@@_notes_bottomrule_bool
3368 {
3369     \IfPackageLoadedTF { booktabs }
3370     {

```

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

```

3371         \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.
3372         { \CT@arc@ \hrule height \heavyrulewidth }
3373     }
3374     { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3375 }
3376 \l_@@_notes_code_after_tl
3377 \seq_gclear:N \g_@@_notes_seq
3378 \seq_gclear:N \g_@@_notes_in_caption_seq
3379 \int_gzero:N \c@tabularnote
3380 }

```

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.

```

3381 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3382 {
3383     \tl_if_novalue:nTF { #1 }
3384     { \item }
3385     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3386 }

```

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`.

```

3387 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3388 {
3389     \pgfpicture
3390     \@@_qpoint:n { row - 1 }
3391     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3392     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3393     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3394     \endpgfpicture
3395     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3396     \int_if_zero:nT \l_@@_first_row_int
3397     {
3398         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3399         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3400     }
3401     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3402 }

```

Now, the general case.

```

3403 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3404 {

```

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

```

3405     \str_if_eq:eeT \l_@@_baseline_tl { t }
3406     { \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`.

```

3407     \pgfpicture
3408     \@@_qpoint:n { row - 1 }

```



```

3409 \dim_gset_eq:Nn \g_tmpa_dim \pgf@y
3410 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3411 {
3412   \int_set:Nn \l_tmpa_int
3413   {
3414     \str_range:Nnn
3415     \l_@@_baseline_tl
3416     { 6 }
3417     { \tl_count:o \l_@@_baseline_tl }
3418   }
3419   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3420 }
3421 {
3422   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3423   \bool_lazy_or:nnT
3424   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3425   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3426   {
3427     \@@_error:n { bad-value-for-baseline }
3428     \int_set:Nn \l_tmpa_int 1
3429   }
3430   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3431 }
3432 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3433 \endpgfpicture
3434 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3435 \int_if_zero:nT \l_@@_first_row_int
3436 {
3437   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3438   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3439 }
3440 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3441 }

```

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.

```

3442 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3443 {

```

We will compute the real width of both delimiters used.

```

3444   \dim_zero_new:N \l_@@_real_left_delim_dim
3445   \dim_zero_new:N \l_@@_real_right_delim_dim
3446   \hbox_set:Nn \l_tmpb_box
3447   {
3448     \m@th
3449     $ % $
3450     \left #1
3451     \vcenter
3452     {
3453       \vbox_to_ht:nn
3454       { \box_ht_plus_dp:N \l_tmpa_box }
3455       { }
3456     }
3457     \right .
3458     $ % $
3459   }
3460   \dim_set:Nn \l_@@_real_left_delim_dim
3461   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3462   \hbox_set:Nn \l_tmpb_box
3463   {
3464     \m@th
3465     $ % $

```

```

3466     \left .
3467     \vbox_to_ht:n {
3468       { \box_ht_plus_dp:N \l_tmpa_box }
3469       { }
3470     \right #2
3471     $ % $
3472   }
3473   \dim_set:Nn \l_@@_real_right_delim_dim
3474   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3475   \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3476   \@@_put_box_in_flow:
3477   \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3478 }

```

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).

```

3479 \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).

```

3480 {
3481   \peek_remove_spaces:n
3482   {
3483     \peek_meaning:NNT \end
3484     \@@_analyze_end:Nn
3485     {
3486       \@@_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`).

```

3487       \@@_array:o \g_@@_array_preamble_tl
3488     }
3489   }
3490 }
3491 {
3492   \@@_create_col_nodes:
3493   \endarray
3494 }

```

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

```

3495 \NewDocumentEnvironment { @@-light-syntax } { b }
3496 {

```

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`.

```

3497   \tl_if_empty:nT { #1 }
3498   { \@@_fatal:n { empty-environment } }
3499   \tl_if_in:nnT { #1 } { & }
3500   { \@@_fatal:n { ampersand-in-light-syntax } }
3501   \tl_if_in:nnT { #1 } { \ }
3502   { \@@_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`.

```
3503 \@@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```
3504 }
```

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.

```
3505 {
3506 \@@_create_col_nodes:
3507 \endarray
3508 }

3509 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3510 {
3511 \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).

```
3512 \seq_clear_new:N \l_@@_rows_seq
```

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

```
3513 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3514 \bool_if:NTF \l_@@_light_syntax_expanded_bool
3515 \seq_set_split:Nee
3516 \seq_set_split:Non
3517 \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

We delete the last row if it is empty.

```
3518 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3519 \tl_if_empty:NF \l_tmpa_tl
3520 { \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.

```
3521 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3522 { \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`.

```
3523 \tl_build_begin:N \l_@@_new_body_tl
3524 \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3525 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3526 \@@_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).

```
3527 \seq_map_inline:Nn \l_@@_rows_seq
3528 {
3529 \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3530 \@@_line_with_light_syntax:n { ##1 }
3531 }
3532 \tl_build_end:N \l_@@_new_body_tl

3533 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3534 {
3535 \int_set:Nn \l_@@_last_col_int
3536 { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3537 }
```

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.

```
3538 \@@_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`).

```
3539 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3540 }
3541 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3542 {
3543   \seq_clear_new:N \l_@@_cells_seq
3544   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3545   \int_set:Nn \l_@@_nb_cols_int
3546     { \int_max:nn \l_@@_nb_cols_int { \seq_count:N \l_@@_cells_seq } }
3547   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3548   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3549   \seq_map_inline:Nn \l_@@_cells_seq
3550     { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3551 }
3552 \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`.

```
3553 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3554 {
3555   \str_if_eq:eeT \g_@@_name_env_str { #2 }
3556     { \@@_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.

```
3557   \end { #2 }
3558 }
```

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`).

```
3559 \cs_new:Npn \@@_create_col_nodes:
3560 {
3561   \crrr
3562   \int_if_zero:nT \l_@@_first_col_int
3563     {
3564       \omit
3565       \hbox_overlap_left:n
3566         {
3567           \bool_if:NT \l_@@_code_before_bool
3568             { \pgfsys@markposition { \@@_env: - col - 0 } }
3569           \pgfpicture
3570             \pgfrememberpicturepositiononpagetrue
3571             \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3572             \str_if_empty:NF \l_@@_name_str
3573               { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3574             \endpgfpicture
3575             \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3576         }
3577       &
3578     }
3579   \omit
```

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

```
3580 \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`).

```

3581 \int_if_zero:nTF \l_@@_first_col_int
3582 {
3583   \@@_mark_position:n { 1 }
3584   \pgfpicture
3585   \pgfrememberpicturepositiononpagetrue
3586   \pgfcoordinate { \@@_env: - col - 1 }
3587   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3588   \str_if_empty:NF \l_@@_name_str
3589   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3590   \endpgfpicture
3591 }
3592 {
3593   \bool_if:NT \l_@@_code_before_bool
3594   {
3595     \hbox
3596     {
3597       \skip_horizontal:n { 0.5 \arrayrulewidth }
3598       \pgfsys@markposition { \@@_env: - col - 1 }
3599       \skip_horizontal:n { -0.5 \arrayrulewidth }
3600     }
3601   }
3602   \pgfpicture
3603   \pgfrememberpicturepositiononpagetrue
3604   \pgfcoordinate { \@@_env: - col - 1 }
3605   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3606   \@@_node_alias:n { 1 }
3607   \endpgfpicture
3608 }

```

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.

```

3609 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3610 \bool_if:NF \l_@@_auto_columns_width_bool
3611 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3612 {
3613   \bool_lazy_and:nnTF
3614   \l_@@_auto_columns_width_bool
3615   { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3616   { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3617   { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3618   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3619 }
3620 \skip_horizontal:N \g_tmpa_skip
3621 \hbox
3622 {
3623   \@@_mark_position:n { 2 }
3624   \pgfpicture
3625   \pgfrememberpicturepositiononpagetrue
3626   \pgfcoordinate { \@@_env: - col - 2 }
3627   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3628   \@@_node_alias:n { 2 }
3629   \endpgfpicture
3630 }

```

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.

```

3631 \int_gset:Nn \g_tmpa_int 1

```

```

3632 \bool_if:NTF \g_@@_last_col_found_bool
3633 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3634 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3635 {
3636     &
3637     \omit
3638     \int_gincr:N \g_tmpa_int

```

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

```

3639     \skip_horizontal:N \g_tmpa_skip
3640     \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

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

```

3641     \pgfpicture
3642     \pgfrememberpicturepositiononpagetrue
3643     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3644     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3645     \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3646     \endpgfpicture
3647 }

```

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).

```

3648 \bool_lazy_or:nnF
3649 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3650 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3651 {
3652     &
3653     \omit
3654     \skip_horizontal:N \g_tmpa_skip
3655     \int_gincr:N \g_tmpa_int
3656     \bool_lazy_any:nF
3657     {
3658         \g_@@_delims_bool
3659         \l_@@_tabular_bool
3660         { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3661         \l_@@_exterior_arraycolsep_bool
3662         \l_@@_bar_at_end_of_pream_bool
3663     }
3664     { \skip_horizontal:n { - \col@sep } }
3665     \bool_if:NT \l_@@_code_before_bool
3666     {
3667         \hbox
3668         {
3669             \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.

```

3670         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3671         { \skip_horizontal:n { - \arraycolsep } }
3672         \pgfsys@markposition
3673         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3674         \skip_horizontal:n { 0.5 \arrayrulewidth }
3675         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3676         { \skip_horizontal:N \arraycolsep }
3677     }
3678 }
3679 \pgfpicture
3680 \pgfrememberpicturepositiononpagetrue
3681 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3682 {
3683     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3684     {
3685         \pgfpoint

```

```

3686         { - 0.5 \arrayrulewidth - \arraycolsep }
3687         \c_zero_dim
3688     }
3689     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3690 }
3691 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3692 \endpgfpicture
3693 }

3694 \bool_if:NT \g_@@_last_col_found_bool
3695 {
3696     \hbox_overlap_right:n
3697     {
3698         \skip_horizontal:N \g_@@_width_last_col_dim
3699         \skip_horizontal:N \colsep
3700         \bool_if:NT \l_@@_code_before_bool
3701         {
3702             \pgfsys@markposition
3703             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3704         }
3705         \pgfpicture
3706         \pgfrememberpicturepositiononpagetrue
3707         \pgfcoordinate
3708         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3709         \pgfpointorigin
3710         \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3711         \endpgfpicture
3712     }
3713 }
3714 }

3715 \cs_new_protected:Npn \@@_mark_position:n #1
3716 {
3717     \bool_if:NT \l_@@_code_before_bool
3718     {
3719         \hbox
3720         {
3721             \skip_horizontal:n { -0.5 \arrayrulewidth }
3722             \pgfsys@markposition { \@@_env: - col - #1 }
3723             \skip_horizontal:n { 0.5 \arrayrulewidth }
3724         }
3725     }
3726 }

3727 \cs_new_protected:Npn \@@_node_alias:n #1
3728 {
3729     \str_if_empty:NF \l_@@_name_str
3730     { \pgfnodealias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3731 }

```

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

```

3732 \tl_const:Nn \c_@@_preamble_first_col_tl
3733 {
3734     >
3735     {

```

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

```

3736     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3737     \bool_gset_true:N \g_@@_after_col_zero_bool
3738     \@@_begin_of_row:

```

```

3739 \hbox_set:Nw \l_@@_cell_box
3740 \@@_math_toggle:
3741 \@@_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”.

```

3742 \int_compare:nNnT \c@iRow > \c_zero_int
3743 {
3744   \bool_lazy_or:nnT
3745     { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3746     { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3747   {
3748     \l_@@_code_for_first_col_tl
3749     \xglobal \colorlet { nicematrix-first-col } { . }
3750   }
3751 }
3752 }

```

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`.

```

3753 l
3754 <
3755 {
3756   \@@_math_toggle:
3757   \hbox_set_end:
3758   \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3759   \@@_adjust_size_box:
3760   \@@_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.

```

3761 \dim_gset:Nn \g_@@_width_first_col_dim
3762 { \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.

```

3763 \hbox_overlap_left:n
3764 {
3765   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3766     \@@_node_cell:
3767     { \box_use_drop:N \l_@@_cell_box }
3768     \skip_horizontal:N \l_@@_left_delim_dim
3769     \skip_horizontal:N \l_@@_left_margin_dim
3770     \skip_horizontal:N \l_@@_extra_left_margin_dim
3771   }
3772   \bool_gset_false:N \g_@@_empty_cell_bool
3773   \skip_horizontal:n { -2 \col@sep }
3774 }
3775 }

```

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

```

3776 \tl_const:Nn \c_@@_preamble_last_col_tl
3777 {
3778   >
3779   {
3780     \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).

```

3781 \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.

```

3782 \bool_gset_true:N \g_@@_last_col_found_bool
3783 \int_gincr:N \c@jCol
3784 \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3785 \hbox_set:Nw \l_@@_cell_box
3786 \@@_math_toggle:
3787 \@@_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”.

```

3788     \int_compare:nNnT \c{iRow} > \c_zero_int
3789     {
3790         \bool_lazy_or:nnT
3791         { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3792         { \int_compare_p:nNn \c{iRow} < \l_@@_last_row_int }
3793         {
3794             \l_@@_code_for_last_col_tl
3795             \xglobal \colorlet { nicematrix-last-col } { . }
3796         }
3797     }
3798 }
3799 1
3800 <
3801 {
3802     \@@_math_toggle:
3803     \hbox_set_end:
3804     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3805     \@@_adjust_size_box:
3806     \@@_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.

```

3807     \dim_gset:Nn \g_@@_width_last_col_dim
3808     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3809     \skip_horizontal:n { -2 \col@sep }

```

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

```

3810     \hbox_overlap_right:n
3811     {
3812         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3813         {
3814             \skip_horizontal:N \l_@@_right_delim_dim
3815             \skip_horizontal:N \l_@@_right_margin_dim
3816             \skip_horizontal:N \l_@@_extra_right_margin_dim
3817             \@@_node_cell:
3818         }
3819     }
3820     \bool_gset_false:N \g_@@_empty_cell_bool
3821 }
3822 }

```

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

```

3823 \NewDocumentEnvironment { NiceArray } { }
3824 {
3825     \bool_gset_false:N \g_@@_delims_bool
3826     \str_if_empty:NT \g_@@_name_env_str
3827     { \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).

```

3828     \NiceArrayWithDelims . .
3829 }
3830 { \endNiceArrayWithDelims }

```

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

```

3831 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3832 {
3833     \NewDocumentEnvironment { #1 NiceArray } { }
3834     {

```

```

3835 \bool_gset_true:N \g_@@_delims_bool
3836 \str_if_empty:NT \g_@@_name_env_str
3837 { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3838 \@@_test_if_math_mode:
3839 \NiceArrayWithDelims #2 #3
3840 }
3841 { \endNiceArrayWithDelims }
3842 }
3843 \@@_def_env:NNN p ( )
3844 \@@_def_env:NNN b [ ]
3845 \@@_def_env:NNN B \{ \}
3846 \@@_def_env:NNN v \vert \vert
3847 \@@_def_env:NNN V \Vert \Vert

```

13 The environment {NiceMatrix} and its variants

13.1 Definition of {pNiceMatrix}

```

3848 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3849 {
3850 \bool_set_false:N \l_@@_preamble_bool
3851 \tl_clear:N \l_tmpa_tl
3852 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3853 { \tl_set:Nn \l_tmpa_tl { @ { } } }
3854 \tl_put_right:Nn \l_tmpa_tl
3855 {
3856 *
3857 {
3858 \int_case:nnF \l_@@_last_col_int
3859 {
3860 { -2 } { \c@MaxMatrixCols }
3861 { -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).

```

3862 }
3863 { \int_eval:n { \l_@@_last_col_int - 1 } }
3864 }
3865 { #2 }
3866 }
3867 \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3868 \exp_args:No \l_tmpb_tl \l_tmpa_tl
3869 }
3870 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3871 \clist_map_inline:nn { p , b , B , v , V }
3872 {
3873 \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3874 {
3875 \bool_gset_true:N \g_@@_delims_bool
3876 \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3877 \int_if_zero:nT \l_@@_last_col_int
3878 {
3879 \bool_set_true:N \l_@@_last_col_without_value_bool
3880 \int_set:Nn \l_@@_last_col_int { -1 }
3881 }
3882 \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3883 \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3884 }
3885 { \use:c { end #1 NiceArray } }
3886 }

```

We define also an environment {NiceMatrix}

```

3887 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3888 {
3889   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3890   \int_if_zero:nT \l_@@_last_col_int
3891   {
3892     \bool_set_true:N \l_@@_last_col_without_value_bool
3893     \int_set:Nn \l_@@_last_col_int { -1 }
3894   }
3895   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3896   \bool_lazy_or:nnT
3897     \l_@@_except_borders_bool
3898     { \clist_if_empty_p:N \l_@@_vlines_clist }
3899     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3900   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3901 }
3902 { \endNiceArray }

```

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

```

3903 \cs_new_protected:Npn \@@_NotEmpty:
3904 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

13.2 The key renew-matrix

```

3905 \cs_set_protected:Npn \@@_renew_matrix:
3906 {
3907   \tl_map_inline:nn { pvVbB }
3908   { \RenewEnvironmentCopy { ##1matrix } { ##1NiceMatrix } }
3909 }

```

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

```

3910 \NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
3911 {

```

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.

```

3912   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3913   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3914   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3915   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3916   \tl_if_empty:NF \l_@@_short_caption_tl
3917   {
3918     \tl_if_empty:NT \l_@@_caption_tl
3919     {
3920       \@@_error_or_warning:n { short-caption-without~caption }
3921       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3922     }
3923   }
3924   \tl_if_empty:NF \l_@@_label_tl
3925   {
3926     \tl_if_empty:NT \l_@@_caption_tl
3927     { \@@_error_or_warning:n { label~without~caption } }
3928   }
3929   \NewDocumentEnvironment { TabularNote } { b }
3930   {
3931     \bool_if:NTF \l_@@_in_code_after_bool
3932     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3933     {
3934       \tl_if_empty:NF \g_@@_tabularnote_tl
3935       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }

```

```

3936         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3937     }
3938 }
3939 { }
3940 \@@_settings_for_tabular:
3941 \NiceArray { #2 }
3942 }
3943 { \endNiceArray }
3944 \cs_new_protected:Npn \@@_settings_for_tabular:
3945 {
3946     \bool_set_true:N \l_@@_tabular_bool
3947     \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3948     \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3949     \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3950 }

3951 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3952 {
3953     \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3954     \dim_set:Nn \l_@@_width_dim { #1 }
3955     \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3956     \@@_settings_for_tabular:
3957     \NiceArray { #3 }
3958 }
3959 {
3960     \endNiceArray
3961     \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3962     { \@@_error:n { NiceTabularX~without~X } }
3963 }

3964 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3965 {
3966     \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3967     \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3968     \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3969     \@@_settings_for_tabular:
3970     \NiceArray { #3 }
3971 }
3972 { \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`).

```

3973 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3974 {
3975     \bool_lazy_all:nT
3976     {
3977         { \dim_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3978         { \l_@@_hvlines_bool }
3979         { ! \g_@@_delims_bool }
3980         { ! \l_@@_except_borders_bool }
3981     }
3982     {
3983         \bool_set_true:N \l_@@_except_borders_bool
3984         \clist_if_empty:NF \l_@@_corners_clist
3985         { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3986         \tl_gput_right:Nn \g_@@_pre_code_after_tl

```

```

3987     {
3988         \@@_stroke_block:nnnnn
3989         {
3990             rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3991             draw = \l_@@_rules_color_tl
3992         }
3993         { 1 } { 1 } { \int_use:N \c@iRow } { \int_use:N \c@jCol }
3994     }
3995 }
3996 }

3997 \cs_new_protected:Npn \@@_after_array:
3998 {

```

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`.

```

3999     \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
4000     \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.

```

4001     \bool_if:NT \g_@@_last_col_found_bool
4002     { \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`.

```

4003     \bool_if:NT \l_@@_last_col_without_value_bool
4004     { \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.

```

4005     \bool_if:NT \l_@@_last_row_without_value_bool
4006     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

4007     \tl_gput_right:Ne \g_@@_aux_tl
4008     {
4009         \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
4010         {
4011             \int_use:N \l_@@_first_row_int ,
4012             \int_use:N \c@iRow ,
4013             \int_use:N \g_@@_row_total_int ,
4014             \int_use:N \l_@@_first_col_int ,
4015             \int_use:N \c@jCol ,
4016             \int_use:N \g_@@_col_total_int
4017         }
4018     }

4019     \clist_if_empty:NF \g_@@_cbic_clist { \@@_cbic: }

```

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`).

```

4020     \seq_if_empty:NF \g_@@_pos_of_blocks_seq
4021     {
4022         \tl_gput_right:Ne \g_@@_aux_tl
4023         {
4024             \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
4025             { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }

```

```

4026     }
4027   }
4028   \seq_if_empty:NF \g_@@_multicolumn_cells_seq
4029   {
4030     \tl_gput_right:Ne \g_@@_aux_tl
4031     {
4032       \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
4033       { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
4034       \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
4035       { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
4036     }
4037   }

```

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

```

4038   \@@_create_diag_nodes:

```

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

```

4039   \pgfpicture
4040   \@@_create_aliases_last:
4041   \str_if_empty:NF \l_@@_name_str { \@@_create_alias_nodes: }
4042   \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.

```

4043   \bool_if:NT \l_@@_parallelize_diags_bool
4044   {
4045     \int_gzero:N \g_@@_ddots_int
4046     \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.

```

4047     \dim_gzero:N \g_@@_delta_x_one_dim
4048     \dim_gzero:N \g_@@_delta_y_one_dim
4049     \dim_gzero:N \g_@@_delta_x_two_dim
4050     \dim_gzero:N \g_@@_delta_y_two_dim
4051   }
4052   \bool_set_false:N \l_@@_initial_open_bool
4053   \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.

```

4054   \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

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

```

4055   \@@_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.

```

4056   \clist_if_empty:NF \l_@@_corners_clist
4057   {
4058     \bool_if:NTF \l_@@_no_cell_nodes_bool
4059     { \@@_error:n { corners-with-no-cell-nodes } }
4060     \@@_compute_corners:
4061   }

```

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

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

```
4062 \seq_gconcat:NNN \g_@@_pos_of_blocks_seq
4063 \g_@@_pos_of_blocks_seq
4064 \g_@@_future_pos_of_blocks_seq
4065 \seq_gclear:N \g_@@_future_pos_of_blocks_seq
```

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

```
4066 \@@_adjust_pos_of_blocks_seq:
4067 \@@_deal_with_rounded_corners:
4068 \clist_if_empty:NF \l_@@_hlines_clist { \@@_draw_hlines: }
4069 \clist_if_empty:NF \l_@@_vlines_clist { \@@_draw_vlines: }
```

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

```
4070 \IfPackageLoadedT { tikz }
4071 {
4072   \tikzset
4073   {
4074     every-picture / .style =
4075     {
4076       overlay ,
4077       remember~picture ,
4078       name~prefix = \@@_env: -
4079     }
4080   }
4081 }
4082 \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
4083 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
4084 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
4085 \cs_set_eq:NN \OverBrace \@@_OverBrace
4086 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
4087 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
4088 \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.

```
4089 \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
4090 \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.

```
4091 \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).

```
4092 \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`).

```
4093 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
4094 { \@@_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:`.

```
4095 \bool_set_true:N \l_@@_in_code_after_bool
4096 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4097 \scan_stop:
4098 \tl_gclear:N \g_nicematrix_code_after_tl
4099 \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.

```

4100   \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
4101   \tl_if_empty:NF \g_@@_pre_code_before_tl
4102   {
4103     \tl_gput_right:Ne \g_@@_aux_tl
4104     {
4105       \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
4106       { \exp_not:o \g_@@_pre_code_before_tl }
4107     }
4108     \tl_gclear:N \g_@@_pre_code_before_tl
4109   }
4110   \tl_if_empty:NF \g_nicematrix_code_before_tl
4111   {
4112     \tl_gput_right:Ne \g_@@_aux_tl
4113     {
4114       \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
4115       { \exp_not:o \g_nicematrix_code_before_tl }
4116     }
4117     \tl_gclear:N \g_nicematrix_code_before_tl
4118   }

4119   \str_gclear:N \g_@@_name_env_str
4120   \@@_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`.

```

4121   \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
4122   }

4123   \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
4124   {
4125     \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
4126     \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }

4127     \dim_set:Nn \l_@@_xdots_shorten_start_dim
4128     { 0.6 \l_@@_xdots_shorten_start_dim }
4129     \dim_set:Nn \l_@@_xdots_shorten_end_dim
4130     { 0.6 \l_@@_xdots_shorten_end_dim }
4131   }

```

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`.

```

4132   \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4133   { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

4134   \cs_new_protected:Npn \@@_create_alias_nodes:
4135   {
4136     \int_step_inline:nn \c@iRow
4137     {
4138       \pgfnodealias
4139       { \l_@@_name_str - ##1 - last }

```

¹³e.g. `\color[rgb]{0.5,0.5,0}`


```

4140     { \@@_env: - ##1 - \int_use:N \c@jCol }
4141   }
4142   \int_step_inline:nn \c@jCol
4143   {
4144     \pgfnodealias
4145       { \l_@@_name_str - last - ##1 }
4146       { \@@_env: - \int_use:N \c@iRow - ##1 }
4147   }
4148   \pgfnodealias
4149     { \l_@@_name_str - last - last }
4150     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
4151 }

```

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).

```

4152 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4153 {
4154   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4155   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4156 }

```

The following command must *not* be protected.

```

4157 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4158 {
4159   { #1 }
4160   { #2 }
4161   {
4162     \int_compare:nNnTF { #3 } > { 98 }
4163     { \int_use:N \c@iRow }
4164     { #3 }
4165   }
4166   {
4167     \int_compare:nNnTF { #4 } > { 98 }
4168     { \int_use:N \c@jCol }
4169     { #4 }
4170   }
4171   { #5 }
4172 }

```

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).

```

4173 \hook_gput_code:nnn { begindocument } { . }
4174 {
4175   \cs_new_protected:Npe \@@_draw_dotted_lines:
4176   {
4177     \c_@@_pgfortikzpicture_tl
4178     \@@_draw_dotted_lines_i:
4179     \c_@@_endpgfortikzpicture_tl
4180   }
4181 }

```

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

```

4182 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4183 {
4184   \pgfrememberpicturepositiononpagetrue

```

```

4185 \pgf@relevantforpicturesizefalse
4186 \g_@@_HVDotsfor_lines_tl
4187 \g_@@_Vdots_lines_tl
4188 \g_@@_Ddots_lines_tl
4189 \g_@@_Iddots_lines_tl
4190 \g_@@_Cdots_lines_tl
4191 \g_@@_Ldots_lines_tl
4192 }

4193 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4194 {
4195   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4196   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4197 }

```

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

```

4198 \pgfdeclareshape { @@_diag_node }
4199 {
4200   \savedanchor { \five }
4201   {
4202     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4203     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4204   }
4205   \anchor { 5 } { \five }
4206   \anchor { center } { \pgfpointorigin }
4207   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4208   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4209   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4210   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4211   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4212   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4213   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4214   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4215   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4216   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4217 }

```

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).

```

4218 \cs_new_protected:Npn \@@_create_diag_nodes:
4219 {
4220   \pgfpicture
4221   \pgfrememberpicturepositiononpagetrue
4222   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4223   {
4224     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4225     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4226     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4227     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4228     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4229     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4230     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4231     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4232     \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.

```

4233   \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4234   \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4235   \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }

```

```

4236 \str_if_empty:NF \l_@@_name_str
4237 { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4238 }

```

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

```

4239 \int_set:Nn \l_tmpa_int { 1 + \int_max:nn \c@iRow \c@jCol } % modified
4240 \@@_qpoint:n { row - \int_min:nn \l_tmpa_int { \c@iRow + 1 } }
4241 \dim_set_eq:NN \l_tmpa_dim \pgf@y
4242 \@@_qpoint:n { col - \int_min:nn \l_tmpa_int { \c@jCol + 1 } }
4243 \pgfcoordinate
4244 { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4245 \pgfnodealias
4246 { \@@_env: - last }
4247 { \@@_env: - \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
4248 \str_if_empty:NF \l_@@_name_str
4249 {
4250 \pgfnodealias
4251 { \l_@@_name_str - \int_use:N \l_tmpa_int }
4252 { \@@_env: - \int_use:N \l_tmpa_int }
4253 \pgfnodealias
4254 { \l_@@_name_str - last }
4255 { \@@_env: - last }
4256 }
4257 \endpgfpicture
4258 }

```

16 We draw the dotted lines

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

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

The command `\@@_find_extremities: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.

We provide first a version in the L3 syntax, and, then a version slightly more efficient.

```

\cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
{
  \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
  \int_set:Nn \l_@@_initial_i_int { #1 }
  \int_set:Nn \l_@@_initial_j_int { #2 }
  \int_set:Nn \l_@@_final_i_int { #1 }
  \int_set:Nn \l_@@_final_j_int { #2 }
  \bool_set_false:N \l_@@_stop_loop_bool
  \bool_do_until:Nn \l_@@_stop_loop_bool
  {
    \int_add:Nn \l_@@_final_i_int { #3 }
    \int_add:Nn \l_@@_final_j_int { #4 }
    \bool_set_false:N \l_@@_final_open_bool
    \int_compare:nNnTF { \l_@@_final_i_int } > { \l_@@_row_max_int }
    {
      \int_compare:nNnTF { #3 } = { 1 }
      { \bool_set_true:N \l_@@_final_open_bool }
      {
        \int_compare:nNnT { \l_@@_final_j_int } > { \l_@@_col_max_int }
        { \bool_set_true:N \l_@@_final_open_bool }
      }
    }
  }
  {
    \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
    {
      \int_compare:nNnT { #4 } = { -1 }
      { \bool_set_true:N \l_@@_final_open_bool }
    }
    {
      \int_compare:nNnT { \l_@@_final_j_int } > { \l_@@_col_max_int }
      {
        \int_compare:nNnT { #4 } = { 1 }
        { \bool_set_true:N \l_@@_final_open_bool }
      }
    }
  }
}
\bool_if:NTF \l_@@_final_open_bool
{
  \int_sub:Nn \l_@@_final_i_int { #3 }
  \int_sub:Nn \l_@@_final_j_int { #4 }
  \bool_set_true:N \l_@@_stop_loop_bool
}
{
  \cs_if_exist:cTF
  {
    @@ _ dotted _
    \int_use:N \l_@@_final_i_int -
    \int_use:N \l_@@_final_j_int
  }
  {
    \int_sub:Nn \l_@@_final_i_int { #3 }
    \int_sub:Nn \l_@@_final_j_int { #4 }
    \bool_set_true:N \l_@@_final_open_bool
    \bool_set_true:N \l_@@_stop_loop_bool
  }
  {
    \cs_if_exist:cTF
    {
      pgf @ sh @ ns @ \@@_env:
      - \int_use:N \l_@@_final_i_int
      - \int_use:N \l_@@_final_j_int
    }
    { \bool_set_true:N \l_@@_stop_loop_bool }
  }
}

```

```

        {
            \cs_set_nopar:cpn
            {
                @@ _ dotted _
                \int_use:N \l_@@_final_i_int -
                \int_use:N \l_@@_final_j_int
            }
            { }
        }
    }
}
}
\bool_set_false:N \l_@@_stop_loop_bool
\int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
\bool_do_until:Nn \l_@@_stop_loop_bool
{
    \int_sub:Nn \l_@@_initial_i_int { #3 }
    \int_sub:Nn \l_@@_initial_j_int { #4 }
    \bool_set_false:N \l_@@_initial_open_bool
    \int_compare:nNnTF { \l_@@_initial_i_int } < { \l_@@_row_min_int }
    {
        \int_compare:nNnTF { #3 } = { 1 }
        { \bool_set_true:N \l_@@_initial_open_bool }
        {
            \int_compare:nNnT { \l_@@_initial_j_int } = { \l_tmpa_int }
            { \bool_set_true:N \l_@@_initial_open_bool }
        }
    }
}
{
    \int_compare:nNnTF { \l_@@_initial_j_int } < { \l_@@_col_min_int }
    {
        \int_compare:nNnT { #4 } = { 1 }
        { \bool_set_true:N \l_@@_initial_open_bool }
    }
    {
        \int_compare:nNnT { \l_@@_initial_j_int } > { \l_@@_col_max_int }
        {
            \int_compare:nNnT { #4 } = { -1 }
            { \bool_set_true:N \l_@@_initial_open_bool }
        }
    }
}
\bool_if:NTF \l_@@_initial_open_bool
{
    \int_add:Nn \l_@@_initial_i_int { #3 }
    \int_add:Nn \l_@@_initial_j_int { #4 }
    \bool_set_true:N \l_@@_stop_loop_bool
}
{
    \cs_if_exist:cTF
    {
        @@ _ dotted _
        \int_use:N \l_@@_initial_i_int -
        \int_use:N \l_@@_initial_j_int
    }
    {
        \int_add:Nn \l_@@_initial_i_int { #3 }
        \int_add:Nn \l_@@_initial_j_int { #4 }
        \bool_set_true:N \l_@@_initial_open_bool
        \bool_set_true:N \l_@@_stop_loop_bool
    }
    {
        \cs_if_exist:cTF

```

```

{
  pgf @ sh @ ns @ \@@_env:
  - \int_use:N \l_@@_initial_i_int
  - \int_use:N \l_@@_initial_j_int
}
{ \bool_set_true:N \l_@@_stop_loop_bool }
{
  \cs_set_nopar:cpn
  {
    @@ _ dotted _
    \int_use:N \l_@@_initial_i_int -
    \int_use:N \l_@@_initial_j_int
  }
  { }
}
}
}
}
\seq_gput_right:Ne \g_@@_pos_of_xdots_seq
{
  { \int_use:N \l_@@_initial_i_int }
  { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
  { \int_use:N \l_@@_final_i_int }
  { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
  { }
}
}
}

```

The following version is slightly more efficient.

```

4259 \cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
4260 {

```

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

```

4261   \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4262   \l_@@_initial_i_int = #1
4263   \l_@@_initial_j_int = #2
4264   \l_@@_final_i_int = #1
4265   \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.

```

4266   \let \l_@@_stop_loop_bool \c_false_bool
4267   \bool_do_until:Nn \l_@@_stop_loop_bool
4268   {

```

We test if we are still in the matrix.

```

4269     \advance \l_@@_final_i_int by #3
4270     \advance \l_@@_final_j_int by #4
4271     \let \l_@@_final_open_bool \c_false_bool
4272     \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4273       \if_int_compare:w #3 = \c_one_int
4274         \let \l_@@_final_open_bool \c_true_bool
4275       \else:
4276         \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4277           \let \l_@@_final_open_bool \c_true_bool
4278       \fi:
4279     \fi:
4280   \else:
4281     \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4282       \if_int_compare:w #4 = -1
4283         \let \l_@@_final_open_bool \c_true_bool

```

```

4284         \fi:
4285     \else:
4286         \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4287             \if_int_compare:w #4 = \c_one_int
4288                 \let \l_@@_final_open_bool \c_true_bool
4289             \fi:
4290         \fi:
4291     \fi:
4292 \fi:
4293 \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.

```

4294 {

```

We do a step backwards.

```

4295     \advance \l_@@_final_i_int by - #3
4296     \advance \l_@@_final_j_int by - #4
4297     \let \l_@@_stop_loop_bool \c_true_bool
4298 }

```

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`.

```

4299 {
4300     \cs_if_exist:cTF
4301     {
4302         @@ _ dotted _
4303         \int_use:N \l_@@_final_i_int -
4304         \int_use:N \l_@@_final_j_int
4305     }
4306     {
4307         \advance \l_@@_final_i_int by - #3
4308         \advance \l_@@_final_j_int by - #4
4309         \let \l_@@_final_open_bool \c_true_bool
4310         \let \l_@@_stop_loop_bool \c_true_bool
4311     }
4312     {
4313         \cs_if_exist:cTF
4314         {
4315             pgf @ sh @ ns @ \@@_env:
4316             - \int_use:N \l_@@_final_i_int
4317             - \int_use:N \l_@@_final_j_int
4318         }
4319         { \let \l_@@_stop_loop_bool \c_true_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.

```

4320     {
4321         \cs_set_nopar:cpn
4322         {
4323             @@ _ dotted _
4324             \int_use:N \l_@@_final_i_int -
4325             \int_use:N \l_@@_final_j_int
4326         }
4327     { }
4328 }
4329 }
4330 }
4331 }

```

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

```
4332 \let \l_@@_stop_loop_bool \c_false_bool
```

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

```
4333 \l_tmpa_int = \l_@@_col_min_int
4334 \advance \l_tmpa_int by -1
4335 \bool_do_until:Nn \l_@@_stop_loop_bool
4336 {
4337   \advance \l_@@_initial_i_int by - #3
4338   \advance \l_@@_initial_j_int by - #4
4339   \let \l_@@_initial_open_bool \c_false_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.

```
4340 \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4341 \if_int_compare:w #3 = \c_one_int
4342 \let \l_@@_initial_open_bool \c_true_bool
4343 \else:
```

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

```
4344 \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4345 \let \l_@@_initial_open_bool \c_true_bool
4346 \fi:
4347 \fi:
4348 \else:
4349 \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4350 \if_int_compare:w #4 = \c_one_int
4351 \let \l_@@_initial_open_bool \c_true_bool
4352 \fi:
4353 \else:
4354 \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4355 \if_int_compare:w #4 = -1
4356 \let \l_@@_initial_open_bool \c_true_bool
4357 \fi:
4358 \fi:
4359 \fi:
4360 \fi:
4361 \bool_if:NTF \l_@@_initial_open_bool
4362 {
4363   \advance \l_@@_initial_i_int by #3
4364   \advance \l_@@_initial_j_int by #4
4365   \let \l_@@_stop_loop_bool \c_true_bool
4366 }
4367 {
4368   \cs_if_exist:cTF
4369   {
4370     @@ _ dotted _
4371     \int_use:N \l_@@_initial_i_int -
4372     \int_use:N \l_@@_initial_j_int
4373   }
4374   {
4375     \advance \l_@@_initial_i_int by #3
4376     \advance \l_@@_initial_j_int by #4
4377     \let \l_@@_initial_open_bool \c_true_bool
4378     \let \l_@@_stop_loop_bool \c_true_bool
4379   }
4380   {
4381     \cs_if_exist:cTF
4382     {
4383       pgf @ sh @ ns @ \@@_env:
4384       - \int_use:N \l_@@_initial_i_int
4385       - \int_use:N \l_@@_initial_j_int
4386     }
4387     { \let \l_@@_stop_loop_bool \c_true_bool }
4388   }
```



```

4389         \cs_set_nopar:cpn
4390         {
4391             @@_dotted_
4392             \int_use:N \l_@@_initial_i_int -
4393             \int_use:N \l_@@_initial_j_int
4394         }
4395         { }
4396     }
4397 }
4398 }
4399 }

```

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.

```

4400     \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4401     {
4402         { \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`.

```

4403         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4404         { \int_use:N \l_@@_final_i_int }
4405         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4406         { }
4407     }
4408 }

```

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.

```

4409 \cs_new_protected:Npn \@@_open_shorten:
4410 {
4411     \bool_if:NT \l_@@_initial_open_bool
4412     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4413     \bool_if:NT \l_@@_final_open_bool
4414     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4415 }

```

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).

```

4416 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4417 {
4418     \int_set_eq:NN \l_@@_row_min_int \c_one_int
4419     \int_set_eq:NN \l_@@_col_min_int \c_one_int
4420     \int_set_eq:NN \l_@@_row_max_int \c_iRow
4421     \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`.

```

4422     \seq_if_empty:NF \g_@@_submatrix_seq
4423     {
4424         \seq_map_inline:Nn \g_@@_submatrix_seq
4425         { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
4426     }
4427 }

```

#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.

```
4428 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4429 {
4430   \if_int_compare:w #3 > #1
4431   \else:
4432     \if_int_compare:w #1 > #5
4433     \else:
4434       \if_int_compare:w #4 > #2
4435       \else:
4436         \if_int_compare:w #2 > #6
4437         \else:
4438           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4439           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4440           \if_int_compare:w \l_@@_row_max_int > #5 \l_@@_row_max_int = #5 \fi:
4441           \if_int_compare:w \l_@@_col_max_int > #6 \l_@@_col_max_int = #6 \fi:
4442         \fi:
4443       \fi:
4444     \fi:
4445   \fi:
4446 }

4447 \cs_new_protected:Npn \@@_set_initial_coords:
4448 {
4449   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4450   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4451 }
4452 \cs_new_protected:Npn \@@_set_final_coords:
4453 {
4454   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4455   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4456 }
4457 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4458 {
4459   \pgfpointanchor
4460   {
4461     \@@_env:
4462     - \int_use:N \l_@@_initial_i_int
4463     - \int_use:N \l_@@_initial_j_int
4464   }
4465   { #1 }
4466   \@@_set_initial_coords:
4467 }
4468 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4469 {
4470   \pgfpointanchor
```

```

4471     {
4472       \@@_env:
4473       - \int_use:N \l_@@_final_i_int
4474       - \int_use:N \l_@@_final_j_int
4475     }
4476     { #1 }
4477     \@@_set_final_coords:
4478   }

4479   \cs_new_protected:Npn \@@_open_x_initial_dim:
4480   {
4481     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4482     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4483     {
4484       \cs_if_exist:cT
4485       { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4486       {
4487         \pgfpointanchor
4488         { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4489         { west }
4490         \dim_set:Nn \l_@@_x_initial_dim
4491         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4492       }
4493     }

```

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

```

4494     \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4495     {
4496       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4497       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4498       \dim_add:Nn \l_@@_x_initial_dim \col@sep
4499     }
4500   }

4501   \cs_new_protected:Npn \@@_open_x_final_dim:
4502   {
4503     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4504     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4505     {
4506       \cs_if_exist:cT
4507       { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4508       {
4509         \pgfpointanchor
4510         { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4511         { east }
4512         \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4513         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4514       }
4515     }

```

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

```

4516     \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4517     {
4518       \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4519       \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4520       \dim_sub:Nn \l_@@_x_final_dim \col@sep
4521     }
4522   }

```

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.

```

4523   \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4524   {
4525     \@@_adjust_to_submatrix:nn { #1 } { #2 }

```

```

4526 \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4527 {
4528 \@@_find_extremities: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.

```

4529 \bool_if:NT \g_@@_aux_found_bool
4530 {
4531 \group_begin:
4532 \@@_open_shorten:
4533 \int_if_zero:nTF { #1 }
4534 { \color { nicematrix-first-row } }
4535 {

```

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.

```

4536 \int_compare:nNnT { #1 } = \l_@@_last_row_int
4537 { \color { nicematrix-last-row } }
4538 }
4539 \keys_set:nn { nicematrix / xdots } { #3 }
4540 \@@_color:o \l_@@_xdots_color_tl
4541 \@@_actually_draw_Ldots:
4542 \group_end:
4543 }
4544 }
4545 }

```

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`.

```

4546 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4547 {
4548 \bool_if:NTF \l_@@_initial_open_bool
4549 {
4550 \@@_open_x_initial_dim:
4551 \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4552 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4553 }
4554 { \@@_set_initial_coords_from_anchor:n { base-east } }
4555 \bool_if:NTF \l_@@_final_open_bool
4556 {
4557 \@@_open_x_final_dim:
4558 \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4559 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4560 }
4561 { \@@_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.

```

4562 \bool_lazy_all:nTF
4563 {
4564 \l_@@_initial_open_bool
4565 \l_@@_final_open_bool

```

```

4566     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4567   }
4568   {
4569     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4570     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4571   }

```

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 (?).

```

4572   {
4573     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4574     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4575   }
4576   \@@_draw_line:
4577 }

```

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.

```

4578 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4579 {
4580   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4581   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4582   {
4583     \@@_find_extremities: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.

```

4584     \bool_if:NT \g_@@_aux_found_bool
4585     {
4586       \group_begin:
4587       \@@_open_shorten:
4588       \int_if_zero:nTF { #1 }
4589       { \color { nicematrix-first-row } }
4590       {

```

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.

```

4591         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4592         { \color { nicematrix-last-row } }
4593       }
4594       \keys_set:nn { nicematrix / xdots } { #3 }
4595       \@@_color:o \l_@@_xdots_color_tl
4596       \@@_actually_draw_Cdots:
4597     \group_end:
4598   }
4599 }
4600 }

```

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`.

```

4601 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4602 {
4603   \bool_if:NTF \l_@@_initial_open_bool
4604     \@@_open_x_initial_dim:
4605     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4606   \bool_if:NTF \l_@@_final_open_bool
4607     \@@_open_x_final_dim:
4608     { \@@_set_final_coords_from_anchor:n { mid-west } }
4609   \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4610   {
4611     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4612     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4613     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4614     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4615     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4616   }
4617   {
4618     \bool_if:NT \l_@@_initial_open_bool
4619     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4620     \bool_if:NT \l_@@_final_open_bool
4621     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4622   }
4623   \@@_draw_line:
4624 }
4625 \cs_new_protected:Npn \@@_open_y_initial_dim:
4626 {
4627   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4628   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4629   {
4630     \cs_if_exist:cT
4631     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4632     {
4633       \pgfpointanchor
4634       { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4635       { north }
4636       \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4637       { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4638     }
4639   }
4640   \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4641   {
4642     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4643     \dim_set:Nn \l_@@_y_initial_dim
4644     {
4645       \fp_to_dim:n
4646       {
4647         \pgf@y
4648         + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4649       }
4650     }
4651   }
4652 }
4653 \cs_new_protected:Npn \@@_open_y_final_dim:
4654 {
4655   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4656   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4657   {
4658     \cs_if_exist:cT
4659     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4660     {
4661       \pgfpointanchor
4662       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4663       { south }

```

```

4664         \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4665         { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4666     }
4667 }
4668 \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4669 {
4670     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4671     \dim_set:Nn \l_@@_y_final_dim
4672         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4673 }
4674 }

```

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.

```

4675 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4676 {
4677     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4678     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4679     {
4680         \@@_find_extremities: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.

```

4681         \bool_if:NT \g_@@_aux_found_bool
4682         {
4683             \group_begin:
4684             \@@_open_shorten:
4685             \int_if_zero:nTF { #2 }
4686             { \color { nicematrix-first-col } }
4687             {
4688                 \int_compare:nNnT { #2 } = \l_@@_last_col_int
4689                 { \color { nicematrix-last-col } }
4690             }
4691             \keys_set:nn { nicematrix / xdots } { #3 }
4692             \@@_color:o \l_@@_xdots_color_tl
4693             \bool_if:NTF \l_@@_Vbrace_bool
4694             \@@_actually_draw_Vbrace:
4695             \@@_actually_draw_Vdots:
4696             \group_end:
4697         }
4698     }
4699 }

```

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`.

```

4700 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4701 {
4702     \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4703     \@@_actually_draw_Vdots_i:
4704     \@@_actually_draw_Vdots_ii:
4705     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4706     \@@_draw_line:
4707 }

```

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

```

4708 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4709 {
4710   \@@_open_y_initial_dim:
4711   \@@_open_y_final_dim:
4712   \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4713 {
4714   \@@_qpoint:n { col - 1 }
4715   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4716   \dim_sub:Nn \l_@@_x_initial_dim
4717   { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4718 }
4719 {
4720   \bool_lazy_and:nnTF
4721   { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4722   { \int_compare_p:nNn \l_@@_initial_j_int = \g_@@_col_total_int }

```

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

```

4723 {
4724   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4725   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4726   \dim_add:Nn \l_@@_x_initial_dim
4727   { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4728 }

```

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

```

4729 {
4730   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4731   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4732   \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4733   \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4734 }
4735 }
4736 }

```

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.

```

4737 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4738 {
4739   \bool_set_false:N \l_tmpa_bool
4740   \bool_if:NF \l_@@_initial_open_bool
4741   {
4742     \bool_if:NF \l_@@_final_open_bool
4743     {
4744       \@@_set_initial_coords_from_anchor:n { south-west }
4745       \@@_set_final_coords_from_anchor:n { north-west }
4746       \bool_set:Nn \l_tmpa_bool
4747       { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4748     }
4749   }

```

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

```

4750   \bool_if:NTF \l_@@_initial_open_bool
4751   {
4752     \@@_open_y_initial_dim:
4753     \@@_set_final_coords_from_anchor:n { north }
4754     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4755   }
4756   {
4757     \@@_set_initial_coords_from_anchor:n { south }

```



```

4758      \bool_if:NTF \l_@@_final_open_bool
4759      \@@_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.

```

4760      {
4761      \@@_set_final_coords_from_anchor:n { north }
4762      \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4763      {
4764      \dim_set:Nn \l_@@_x_initial_dim
4765      {
4766      \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4767      \l_@@_x_initial_dim \l_@@_x_final_dim
4768      }
4769      }
4770      }
4771  }
4772 }

```

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`.

```

4773 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4774 {
4775   \bool_if:NTF \l_@@_initial_open_bool
4776   \@@_open_y_initial_dim:
4777   { \@@_set_initial_coords_from_anchor:n { south } }
4778   \bool_if:NTF \l_@@_final_open_bool
4779   \@@_open_y_final_dim:
4780   { \@@_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.

```

4781   \int_if_zero:nTF \l_@@_initial_j_int
4782   {
4783     \@@_qpoint:n { col - 1 }
4784     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4785     \dim_sub:Nn \l_@@_x_initial_dim
4786     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4787   }

```

Elsewhere, the brace must be drawn left flush.

```

4788   {
4789     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4790     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4791     \dim_add:Nn \l_@@_x_initial_dim
4792     { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4793   }

```

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

```

4794   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4795   \@@_draw_line:
4796 }

```

```

4797 \cs_new:Npn \@@_colsep:
4798 { \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.

```

4799 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4800 {
4801   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4802   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4803   {
4804     \@@_find_extremities: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.

```

4805     \bool_if:NT \g_@@_aux_found_bool
4806     {
4807       \group_begin:
4808       \@@_open_shorten:
4809       \keys_set:nn { nicematrix / xdots } { #3 }
4810       \@@_color:o \l_@@_xdots_color_tl
4811       \@@_actually_draw_Ddots:
4812       \group_end:
4813     }
4814   }
4815 }

```

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`.

```

4816 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4817 {
4818   \bool_if:NTF \l_@@_initial_open_bool
4819   {
4820     \@@_open_y_initial_dim:
4821     \@@_open_x_initial_dim:
4822   }
4823   { \@@_set_initial_coords_from_anchor:n { south~east } }
4824   \bool_if:NTF \l_@@_final_open_bool
4825   {
4826     \@@_open_x_final_dim:
4827     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4828   }
4829   { \@@_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.

```

4830   \bool_if:NT \l_@@_parallelize_diags_bool
4831   {
4832     \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).

```
4833 \int_compare:nNnTF \g_@@_ddots_int = 1
```

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.

```
4834 {
4835   \dim_gset:Nn \g_@@_delta_x_one_dim
4836   { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4837   \dim_gset:Nn \g_@@_delta_y_one_dim
4838   { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4839 }
```

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`.

```
4840 {
4841   \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4842   {
4843     \dim_set:Nn \l_@@_y_final_dim
4844     {
4845       \l_@@_y_initial_dim +
4846       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4847       \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4848     }
4849   }
4850 }
4851 }
4852 \@@_draw_line:
4853 }
```

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.

```
4854 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4855 {
4856   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4857   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4858   {
4859     \@@_find_extremities: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.

```
4860   \bool_if:NT \g_@@_aux_found_bool
4861   {
4862     \group_begin:
4863     \@@_open_shorten:
4864     \keys_set:nn { nicematrix / xdots } { #3 }
4865     \@@_color:o \l_@@_xdots_color_tl
4866     \@@_actually_draw_Iddots:
4867     \group_end:
4868   }
4869 }
4870 }
```

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`.

```

4871 \cs_new_protected:Npn \@@_actually_draw_iddots:
4872 {
4873   \bool_if:NTF \l_@@_initial_open_bool
4874   {
4875     \@@_open_y_initial_dim:
4876     \@@_open_x_initial_dim:
4877   }
4878   { \@@_set_initial_coords_from_anchor:n { south-west } }
4879   \bool_if:NTF \l_@@_final_open_bool
4880   {
4881     \@@_open_y_final_dim:
4882     \@@_open_x_final_dim:
4883   }
4884   { \@@_set_final_coords_from_anchor:n { north-east } }
4885   \bool_if:NT \l_@@_parallelize_diags_bool
4886   {
4887     \int_gincr:N \g_@@_iddots_int
4888     \int_compare:nNnTF \g_@@_iddots_int = 1
4889     {
4890       \dim_gset:Nn \g_@@_delta_x_two_dim
4891       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4892       \dim_gset:Nn \g_@@_delta_y_two_dim
4893       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4894     }
4895     {
4896       \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4897       {
4898         \dim_set:Nn \l_@@_y_final_dim
4899         {
4900           \l_@@_y_initial_dim +
4901           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4902           \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4903         }
4904       }
4905     }
4906   }
4907   \@@_draw_line:
4908 }

```

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`

```

4909 \cs_new_protected:Npn \@@_draw_line:
4910 {
4911   \pgfrememberpicturepositiononpagetrue
4912   \pgf@relevantforpicturesizefalse
4913   \bool_lazy_or:nnTF
4914     \l_@@_dotted_bool
4915     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4916     \@@_draw_standard_dotted_line:
4917     \@@_draw_unstandard_dotted_line:
4918 }

```

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.

```

4919 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4920 {
4921   \begin { scope }
4922     \@@_draw_unstandard_dotted_line:o
4923     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4924 }

```

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.

```

4925 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4926 {
4927   \@@_draw_unstandard_dotted_line:nooo
4928   { #1 }
4929   \l_@@_xdots_up_tl
4930   \l_@@_xdots_down_tl
4931   \l_@@_xdots_middle_tl
4932 }
4933 \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.

```

4934 \hook_gput_code:nnn { begindocument } { . }
4935 {
4936   \IfPackageLoadedT { tikz }
4937   {
4938     \tikzset
4939     {
4940       @@_node_above / .style = { sloped , above } ,
4941       @@_node_below / .style = { sloped , below } ,
4942       @@_node_middle / .style =
4943       {
4944         sloped ,
4945         inner-sep = \c_@@_innersep_middle_dim
4946       }
4947     }
4948   }
4949 }

```

```

4950 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4951 {

```

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.

```

4952   \dim_zero_new:N \l_@@_l_dim

```

```

4953 \dim_set:Nn \l_@@_l_dim
4954 {
4955   \fp_to_dim:n
4956   {
4957     sqrt
4958     (
4959       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4960       +
4961       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4962     )
4963   }
4964 }

```

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.

```

4965 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4966 {
4967   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4968   \@@_draw_unstandard_dotted_line_i:
4969 }

```

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

```

4970 \bool_if:NT \l_@@_xdots_h_labels_bool
4971 {
4972   \tikzset
4973   {
4974     @@_node_above / .style = { auto = left } ,
4975     @@_node_below / .style = { auto = right } ,
4976     @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4977   }
4978 }
4979 \tl_if_empty:nF { #4 }
4980 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4981 \dim_zero:N \l_tmpa_dim
4982 \dim_zero:N \l_tmpb_dim
4983 \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
4984 {

```

We test whether the brace is vertical or horizontal.

```

4985   \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4986   { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
4987   { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
4988 }
4989 {
4990   \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
4991   {
4992     \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4993     { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
4994     { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
4995   }
4996 }
4997 \use:e
4998 {
4999   \exp_not:N \begin { scope }
5000   [ shift = {(\dim_use:N \l_tmpa_dim, \dim_use:N \l_tmpb_dim)} ]
5001 }
5002 \draw
5003 [ #1 ]
5004 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
5005 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
5006 node [ @@_node_below ] { $ \scriptstyle #3 $ }
5007 node [ @@_node_above ] { $ \scriptstyle #2 $ }

```

```

5008         ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
5009     \end { scope }
5010 \end { scope }
5011 }
5012 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
5013 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
5014 {
5015     \dim_set:Nn \l_tmpa_dim
5016     {
5017         \l_@@_x_initial_dim
5018         + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5019         * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
5020     }
5021     \dim_set:Nn \l_tmpb_dim
5022     {
5023         \l_@@_y_initial_dim
5024         + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5025         * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
5026     }
5027     \dim_set:Nn \l_@@_tmpc_dim
5028     {
5029         \l_@@_x_final_dim
5030         - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5031         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5032     }
5033     \dim_set:Nn \l_@@_tmpd_dim
5034     {
5035         \l_@@_y_final_dim
5036         - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5037         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5038     }
5039     \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
5040     \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
5041     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
5042     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
5043 }

```

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

```

5044 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
5045 {
5046     \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.

```

5047     \dim_zero_new:N \l_@@_l_dim
5048     \dim_set:Nn \l_@@_l_dim
5049     {
5050         \fp_to_dim:n
5051         {
5052             sqrt
5053             (
5054                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5055                 +
5056                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5057             )
5058         }
5059     }

```

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.

```

5060 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
5061 {
5062     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
5063     { \@@_draw_standard_dotted_line_i: }
5064 }
5065 \group_end:
5066 \bool_lazy_all:nF
5067 {
5068     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
5069     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
5070     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
5071 }
5072 { \@@_labels_standard_dotted_line: }
5073 }
5074 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
5075 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
5076 {

```

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

```

5077 \int_set:Nn \l_tmpa_int
5078 {
5079     \dim_ratio:nn
5080     {
5081         \l_@@_l_dim
5082         - \l_@@_xdots_shorten_start_dim
5083         - \l_@@_xdots_shorten_end_dim
5084     }
5085     \l_@@_xdots_inter_dim
5086 }

```

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

```

5087 \dim_set:Nn \l_tmpa_dim
5088 {
5089     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5090     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5091 }
5092 \dim_set:Nn \l_tmpb_dim
5093 {
5094     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5095     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5096 }

```

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.

```

5097 \dim_gadd:Nn \l_@@_x_initial_dim
5098 {
5099     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5100     \dim_ratio:nn
5101     {
5102         \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5103         + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5104     }
5105     { 2 \l_@@_l_dim }
5106 }
5107 \dim_gadd:Nn \l_@@_y_initial_dim
5108 {
5109     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5110     \dim_ratio:nn
5111     {
5112         \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5113         + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim

```



```

5114     }
5115     { 2 \l_@@_l_dim }
5116 }
5117 \pgf@relevantforpicturesizefalse
5118 \int_step_inline:nnn \c_zero_int \l_tmpa_int
5119 {
5120     \pgfpathcircle
5121     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5122     \l_@@_xdots_radius_dim
5123     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
5124     \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
5125 }
5126 \pgfusepathqfill
5127 }

5128 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
5129 {
5130     \pgfscope
5131     \pgftransformshift
5132     {
5133         \pgfpointlineattime { 0.5 }
5134         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5135         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
5136     }
5137     \fp_set:Nn \l_tmpa_fp
5138     {
5139         atand
5140         (
5141             \l_@@_y_final_dim - \l_@@_y_initial_dim ,
5142             \l_@@_x_final_dim - \l_@@_x_initial_dim
5143         )
5144     }
5145     \pgftransformrotate { \fp_use:N \l_tmpa_fp }
5146     \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
5147     \tl_if_empty:NF \l_@@_xdots_middle_tl
5148     {
5149         \begin { pgfscope }
5150         \pgfset { inner~sep = \c_@@_innersep_middle_dim }
5151         \pgfnode
5152         { rectangle }
5153         { center }
5154         {
5155             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5156             {
5157                 $ % $
5158                 \scriptstyle \l_@@_xdots_middle_tl
5159                 $ % $
5160             }
5161         }
5162         { }
5163         {
5164             \pgfsetfillcolor { white }
5165             \pgfusepath { fill }
5166         }
5167         \end { pgfscope }
5168     }
5169     \tl_if_empty:NF \l_@@_xdots_up_tl
5170     {
5171         \pgfnode
5172         { rectangle }
5173         { south }
5174         {
5175             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }

```

```

5176         {
5177             $ % $
5178             \scriptstyle \l_@@_xdots_up_tl
5179             $ % $
5180         }
5181     }
5182     { }
5183     { \pgfusepath { } }
5184 }
5185 \tl_if_empty:NF \l_@@_xdots_down_tl
5186 {
5187     \pgfnode
5188     { rectangle }
5189     { north }
5190     {
5191         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5192         {
5193             $ % $
5194             \scriptstyle \l_@@_xdots_down_tl
5195             $ % $
5196         }
5197     }
5198     { }
5199     { \pgfusepath { } }
5200 }
5201 \endpgfscope
5202 }

```

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.

```

5203 \hook_gput_code:nnn { begindocument } { . }
5204 {

```

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

```

5205     \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
5206     \cs_new_protected:Npn \@@_Ldots: { \@@_collect_options:n { \@@_Ldots_i } }
5207     \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5208     {
5209         \int_if_zero:nTF \c@jCol
5210         { \@@_error:nn { in~first~col } { \Ldots } }
5211         {
5212             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5213             { \@@_error:nn { in~last~col } { \Ldots } }
5214             {
5215                 \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5216                 { #1 , down = #2 , up = #3 , middle = #4 }
5217             }
5218         }
5219     }

```

```

5220     { \phantom { \ensuremath { \@@_old_ldots: } } }
5221     \bool_gset_true:N \g_@@_empty_cell_bool
5222 }

5223 \cs_new_protected:Npn \@@_Cdots: { \@@_collect_options:n { \@@_Cdots_i } }
5224 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5225 {
5226     \int_if_zero:nTF \c@jCol
5227     { \@@_error:nn { in~first~col } { \Cdots } }
5228     {
5229         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5230         { \@@_error:nn { in~last~col } { \Cdots } }
5231         {
5232             \@@_instruction_of_type:nnn { \c_false_bool } { Cdots }
5233             { #1 , down = #2 , up = #3 , middle = #4 }
5234         }
5235     }
5236     \bool_if:NF \l_@@_nullify_dots_bool
5237     { \phantom { \ensuremath { \@@_old_cdots: } } }
5238     \bool_gset_true:N \g_@@_empty_cell_bool
5239 }

5240 \cs_new_protected:Npn \@@_Vdots: { \@@_collect_options:n { \@@_Vdots_i } }
5241 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5242 {
5243     \int_if_zero:nTF \c@iRow
5244     { \@@_error:nn { in~first~row } { \Vdots } }
5245     {
5246         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5247         { \@@_error:nn { in~last~row } { \Vdots } }
5248         {
5249             \@@_instruction_of_type:nnn { \c_false_bool } { Vdots }
5250             { #1 , down = #2 , up = #3 , middle = #4 }
5251         }
5252     }
5253     \bool_if:NF \l_@@_nullify_dots_bool
5254     { \phantom { \ensuremath { \@@_old_vdots: } } }
5255     \bool_gset_true:N \g_@@_empty_cell_bool
5256 }

5257 \cs_new_protected:Npn \@@_Ddots: { \@@_collect_options:n { \@@_Ddots_i } }
5258 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5259 {
5260     \int_case:nnF \c@iRow
5261     {
5262         0 { \@@_error:nn { in~first~row } { \Ddots } }
5263         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5264     }
5265     {
5266         \int_case:nnF \c@jCol
5267         {
5268             0 { \@@_error:nn { in~first~col } { \Ddots } }
5269             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5270         }
5271         {
5272             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5273             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5274             { #1 , down = #2 , up = #3 , middle = #4 }
5275         }
5276     }
5277 }

```

```

5278 \bool_if:NF \l_@@_nullify_dots_bool
5279 { \phantom { \ensuremath { \@@_old_ddots: } } }
5280 \bool_gset_true:N \g_@@_empty_cell_bool
5281 }

5282 \cs_new_protected:Npn \@@_Iddots:
5283 { \@@_collect_options:n { \@@_Iddots_i } }
5284 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5285 {
5286 \int_case:nnF \c@iRow
5287 {
5288 0 { \@@_error:nn { in~first~row } { \Iddots } }
5289 \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5290 }
5291 {
5292 \int_case:nnF \c@jCol
5293 {
5294 0 { \@@_error:nn { in~first~col } { \Iddots } }
5295 \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5296 }
5297 {
5298 \keys_set_known:nn { nicematrix / Ddots } { #1 }
5299 \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { Iddots }
5300 { #1 , down = #2 , up = #3 , middle = #4 }
5301 }
5302 }
5303 \bool_if:NF \l_@@_nullify_dots_bool
5304 { \phantom { \ensuremath { \@@_old_iddots: } } }
5305 \bool_gset_true:N \g_@@_empty_cell_bool
5306 }
5307 }

```

End of the \AddToHook.

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

```

5308 \keys_define:nn { nicematrix / Ddots }
5309 {
5310 draw-first .bool_set:N = \l_@@_draw_first_bool ,
5311 draw-first .default:n = true ,
5312 draw-first .value_forbidden:n = true
5313 }

```

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

```

5314 \cs_new_protected:Npn \@@_Hspace:
5315 {
5316 \bool_gset_true:N \g_@@_empty_cell_bool
5317 \hspace
5318 }

```

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.

```

5319 \cs_new_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.

```

5320 \cs_new:Npn \@@_Hdotsfor:
5321 {
5322 \bool_lazy_and:nnTF
5323 { \int_if_zero_p:n \c@jCol }

```

```

5324 { \int_if_zero_p:n \l_@@_first_col_int }
5325 {
5326   \bool_if:NTF \g_@@_after_col_zero_bool
5327   {
5328     \multicolumn { 1 } { c } { }
5329     \@@_Hdotsfor_i:
5330   }
5331   { \@@_fatal:n { Hdotsfor~in~col~0 } }
5332 }
5333 {
5334   \multicolumn { 1 } { c } { }
5335   \@@_Hdotsfor_i:
5336 }
5337 }

```

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:`).

```

5338 \hook_gput_code:nnn { begindocument } { . }
5339 {

```

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

```

5340   \cs_new_protected:Npn \@@_Hdotsfor_i:
5341   { \@@_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`).

```

5342   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5343   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5344   {
5345     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5346     {
5347       \@@_Hdotsfor:nnnn
5348       { \int_use:N \c{iRow }
5349         { \int_use:N \c{jCol }
5350           { #2 }
5351           {
5352             #1 , #3 ,
5353             down = \exp_not:n { #4 } ,
5354             up = \exp_not:n { #5 } ,
5355             middle = \exp_not:n { #6 }
5356           }
5357         }
5358       \prg_replicate:nn { #2 - 1 }
5359       {
5360         &
5361         \multicolumn { 1 } { c } { }
5362         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5363       }
5364     }
5365   }

```

```

5366 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5367 {
5368   \bool_set_false:N \l_@@_initial_open_bool
5369   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5370   \int_set:Nn \l_@@_initial_i_int { #1 }
5371   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5372   \int_compare:nNnTF { #2 } = 1
5373   {

```

```

5374 \int_set:Nn \l_@@_initial_j_int 1
5375 \bool_set_true:N \l_@@_initial_open_bool
5376 }
5377 {
5378 \cs_if_exist:cTF
5379 {
5380 pgf @ sh @ ns @ \@@_env:
5381 - \int_use:N \l_@@_initial_i_int
5382 - \int_eval:n { #2 - 1 }
5383 }
5384 { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5385 {
5386 \int_set:Nn \l_@@_initial_j_int { #2 }
5387 \bool_set_true:N \l_@@_initial_open_bool
5388 }
5389 }
5390 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5391 {
5392 \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5393 \bool_set_true:N \l_@@_final_open_bool
5394 }
5395 {
5396 \cs_if_exist:cTF
5397 {
5398 pgf @ sh @ ns @ \@@_env:
5399 - \int_use:N \l_@@_final_i_int
5400 - \int_eval:n { #2 + #3 }
5401 }
5402 { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5403 {
5404 \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5405 \bool_set_true:N \l_@@_final_open_bool
5406 }
5407 }
5408 \bool_if:NT \g_@@_aux_found_bool
5409 {
5410 \group_begin:
5411 \@@_open_shorten:
5412 \int_if_zero:nTF { #1 }
5413 { \color { nicematrix-first-row } }
5414 {
5415 \int_compare:nNnT { #1 } = \g_@@_row_total_int
5416 { \color { nicematrix-last-row } }
5417 }
5418 \keys_set:nn { nicematrix / xdots } { #4 }
5419 \@@_color:o \l_@@_xdots_color_tl
5420 \@@_actually_draw_Ldots:
5421 \group_end:
5422 }

```

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:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5423 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5424 { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5425 }

5426 \hook_gput_code:nnn { begindocument } { . }
5427 {
5428 \cs_new_protected:Npn \@@_Vdotsfor:
5429 { \@@_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`).

```

5430 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5431 \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5432 {
5433   \bool_gset_true:N \g_@@_empty_cell_bool
5434   \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5435   {
5436     \@@_Vdotsfor:nnnn
5437     { \int_use:N \c@iRow }
5438     { \int_use:N \c@jCol }
5439     { #2 }
5440     {
5441       #1 , #3 ,
5442       down = \exp_not:n { #4 } ,
5443       up = \exp_not:n { #5 } ,
5444       middle = \exp_not:n { #6 }
5445     }
5446   }
5447 }
5448 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5449 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5450 {
5451   \bool_set_false:N \l_@@_initial_open_bool
5452   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5453 \int_set:Nn \l_@@_initial_j_int { #2 }
5454 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5455 \int_compare:nNnTF { #1 } = 1
5456 {
5457   \int_set:Nn \l_@@_initial_i_int 1
5458   \bool_set_true:N \l_@@_initial_open_bool
5459 }
5460 {
5461   \cs_if_exist:cTF
5462   {
5463     pgf @ sh @ ns @ \@@_env:
5464     - \int_eval:n { #1 - 1 }
5465     - \int_use:N \l_@@_initial_j_int
5466   }
5467   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5468   {
5469     \int_set:Nn \l_@@_initial_i_int { #1 }
5470     \bool_set_true:N \l_@@_initial_open_bool
5471   }
5472 }
5473 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5474 {
5475   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5476   \bool_set_true:N \l_@@_final_open_bool
5477 }
5478 {
5479   \cs_if_exist:cTF
5480   {
5481     pgf @ sh @ ns @ \@@_env:
5482     - \int_eval:n { #1 + #3 }

```

```

5483         - \int_use:N \l_@@_final_j_int
5484     }
5485     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5486     {
5487         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5488         \bool_set_true:N \l_@@_final_open_bool
5489     }
5490 }
5491 \bool_if:NT \g_@@_aux_found_bool
5492 {
5493     \group_begin:
5494     \@@_open_shorten:
5495     \int_if_zero:nTF { #2 }
5496     { \color { nicematrix-first-col } }
5497     {
5498         \int_compare:nNnT { #2 } = \g_@@_col_total_int
5499         { \color { nicematrix-last-col } }
5500     }
5501     \keys_set:nn { nicematrix / xdots } { #4 }
5502     \@@_color:o \l_@@_xdots_color_tl
5503     \bool_if:NTF \l_@@_Vbrace_bool
5504     \@@_actually_draw_Vbrace:
5505     \@@_actually_draw_Vdots:
5506     \group_end:
5507 }

```

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:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5508 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5509 { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5510 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5511 \NewDocumentCommand \@@_rotate: { 0 { } }
5512 {
5513     \bool_gset_true:N \g_@@_rotate_bool
5514     \keys_set:nn { nicematrix / rotate } { #1 }
5515     \ignorespaces
5516 }

```

The command `\@@_rotate_p_col:` will be linked to `\rotate` in the the cells of the columns of type *p* and *al*.

```

5517 \cs_new_protected:Npn \@@_rotate_p_col: { \@@_error:n { rotate~in~p~col } }

5518 \keys_define:nn { nicematrix / rotate }
5519 {
5520     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5521     c .value_forbidden:n = true ,
5522     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5523 }

```


19 The command `\line` accessible in `\CodeAfter`

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

```

5524 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5525 {
5526   \tl_if_empty:nTF { #2 }
5527   { #1 }
5528   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5529 }
5530 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5531 { \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).

```

5532 \hook_gput_code:nnn { begindocument } { . }
5533 {
```

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

```

5534   \tl_set_rescan:Nnn \l_tmpa_tl { }
5535   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5536   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5537   {
5538     \group_begin:
5539     \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5540     \@@_color:o \l_@@_xdots_color_tl
5541     \use:e
5542     {
5543       \@@_line_i:nn
5544       { \@@_double_int_eval:n #2 - \q_stop }
5545       { \@@_double_int_eval:n #3 - \q_stop }
5546     }
5547     \group_end:
5548   }
5549 }

5550 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5551 {
5552   \bool_set_false:N \l_@@_initial_open_bool
5553   \bool_set_false:N \l_@@_final_open_bool
5554   \bool_lazy_or:nnTF
5555   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5556   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5557   { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }
```

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

```

5558   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5559 }
```

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

```

5560 \hook_gput_code:nnn { begindocument } { . }
5561 {
5562   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5563   {

```

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:`.

```

5564     \c_@@_pgfortikzpicture_tl
5565     \@@_draw_line_iii:nn { #1 } { #2 }
5566     \c_@@_endpgfortikzpicture_tl
5567   }
5568 }

```

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

```

5569 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5570 {
5571   \pgfrememberpicturepositiononpagetrue
5572   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5573   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5574   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5575   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5576   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5577   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5578   \@@_draw_line:
5579 }

```

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.

```

5580 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNt \c@iRow < }
5581 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNf \c@jCol < }

```

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

```

5582 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5583 {
5584   \tl_gput_right:Ne \g_@@_row_style_tl
5585   {

```

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

```

5586     \exp_not:N
5587     \@@_if_row_less_than:nn
5588     { \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`).

```

5589     {
5590         \exp_not:N
5591         \@@_if_col_greater_than:nn
5592         { \int_use:N \c@jCol }
5593         { \exp_not:n { #1 } \scan_stop: }
5594     }
5595 }
5596 }
5597 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

5598 \keys_define:nn { nicematrix / RowStyle }
5599 {
5600     cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5601     cell-space-top-limit .value_required:n = true ,
5602     cell-space-top-limit+ .code:n =
5603         \dim_set:Nn \l_tmpa_dim { \l_@@_cell_space_top_limit_dim + #1 } ,
5604     cell-space-top-limit+ .value_required:n = true ,
5605     cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
5606     cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5607     cell-space-bottom-limit .value_required:n = true ,
5608     cell-space-bottom-limit+ .code:n =
5609         \dim_set:Nn \l_tmpb_dim { \l_@@_cell_space_bottom_limit_dim + #1 } ,
5610     cell-space-bottom-limit+ .value_required:n = true ,
5611     cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
5612     cell-space-limits .meta:n =
5613     {
5614         cell-space-top-limit = #1 ,
5615         cell-space-bottom-limit = #1 ,
5616     } ,
5617     cell-space-limits+ .meta:n =
5618     {
5619         cell-space-top-limit += #1 ,
5620         cell-space-bottom-limit += #1 ,
5621     } ,
5622     cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
5623     color .tl_set:N = \l_@@_color_tl ,
5624     color .value_required:n = true ,
5625     bold .bool_set:N = \l_@@_bold_row_style_bool ,
5626     bold .default:n = true ,
5627     nb-rows .code:n =
5628         \str_if_eq:eeTF { #1 } { * }
5629         { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5630         { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5631     nb-rows .value_required:n = true ,
5632     fill .tl_set:N = \l_@@_fill_tl ,
5633     fill .value_required:n = true ,

```

In fine, the opacity will be applied by `\pgfsetfillopacity`.

```

5634     opacity .tl_set:N = \l_@@_opacity_tl ,
5635     opacity .value_required:n = true ,
5636     rowcolor .tl_set:N = \l_@@_fill_tl ,
5637     rowcolor .value_required:n = true ,
5638     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5639     rounded-corners .default:n = 4 pt ,
5640     unknown .code:n =
5641         \@@_unknown_key:nn
5642         { nicematrix / RowStyle }
5643         { Unknown-key-for-RowStyle }
5644 }

```

```

5645 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }

```

```

5646 {
5647   \group_begin:
5648   \tl_clear:N \l_@@_fill_tl
5649   \tl_clear:N \l_@@_opacity_tl
5650   \tl_clear:N \l_@@_color_tl
5651   \int_set:Nn \l_@@_key_nb_rows_int 1
5652   \dim_zero:N \l_@@_rounded_corners_dim
5653   \dim_zero:N \l_tmpa_dim
5654   \dim_zero:N \l_tmpb_dim
5655   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5656   \tl_if_empty:NF \l_@@_fill_tl
5657   {
5658     \@@_add_opacity_to_fill:
5659     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5660     {

```

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

```

5661       \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5662       { \int_use:N \c@iRow - \int_use:N \c@jCol }
5663       {
5664         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5665         - *
5666       }
5667       { \dim_use:N \l_@@_rounded_corners_dim }
5668     }
5669   }
5670   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5671   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5672   {
5673     \@@_put_in_row_style:e
5674     {
5675       \@@_put_in_cell_after_hook:n
5676       {

```

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

```

5677       \dim_set:Nn \l_@@_cell_space_top_limit_dim
5678       { \dim_use:N \l_tmpa_dim }
5679     }
5680   }
5681 }

```

\l_tmpb_dim is the value of the key cell-space-bottom-limit of \RowStyle.

```

5682   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5683   {
5684     \@@_put_in_row_style:e
5685     {
5686       \@@_put_in_cell_after_hook:n
5687       {
5688         \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5689         { \dim_use:N \l_tmpb_dim }
5690       }
5691     }
5692   }

```

\l_@@_color_tl is the value of the key color of \RowStyle.

```

5693   \tl_if_empty:NF \l_@@_color_tl
5694   {
5695     \@@_put_in_row_style:e
5696     {
5697       \mode_leave_vertical:
5698       \@@_color:n { \l_@@_color_tl }
5699     }

```

```

5700     }
\l_@@_bold_row_style_bool is the value of the key bold.
5701     \bool_if:NT \l_@@_bold_row_style_bool
5702     {
5703         \@@_put_in_row_style:n
5704         {
5705             \exp_not:n
5706             {
5707                 \if_mode_math:
5708                     $ % $
5709                     \bfseries \boldmath
5710                     $ % $
5711                 \else:
5712                     \bfseries \boldmath
5713                 \fi:
5714             }
5715         }
5716     }
5717 \group_end:
5718 \g_@@_row_style_tl
5719 \ignorespaces
5720 }

```

The following commande must *not* be protected.

```

5721 \cs_new:Npn \@@_rounded_from_row:n #1
5722 {
5723     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

In the following code, the “- 1” is *not* a subtraction.

```

5724     { \int_eval:n { #1 } - 1 }
5725     {
5726         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5727         - \exp_not:n { \int_use:N \c@jCol }
5728     }
5729     { \dim_use:N \l_@@_rounded_corners_dim }
5730 }

```

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).

```
5731 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5732 {
```

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.

```
5733 \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`.

```
5734 \str_if_in:nnF { #1 } { !! }
5735 {
5736 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```
5737 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5738 }
5739 \int_if_zero:nTF \l_tmpa_int
```

First, the case where the color is a *new* color (not in the sequence).

```
5740 {
5741 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5742 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5743 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5744 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5745 }
5746 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5747 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

The following command must be used within a `\pgfpicture`.

```
5748 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5749 {
5750 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5751 {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```
5752 \group_begin:
5753 \pgfsetcornersarced
5754 { \pgfpoint \l_@@_tab_rounded_corners_dim \l_@@_tab_rounded_corners_dim }
```

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.

```
5755 \bool_if:NTF \l_@@_hvlines_bool
5756 {
5757 \pgfpathrectanglecorners
5758 {
5759 \pgfpointadd
5760 { \@@_qpoint:n { row-1 } }
5761 { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
5762 }
5763 {
5764 \pgfpointadd
5765 {
5766 \@@_qpoint:n
5767 { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
5768 }
5769 { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5770 }
```

```

5771     }
5772     {
5773         \pgfpathrectanglecorners
5774         { \@@_qpoint:n { row-1 } }
5775         {
5776             \pgfpointadd
5777             {
5778                 \@@_qpoint:n
5779                 { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
5780             }
5781             { \pgfpoint \c_zero_dim \arrayrulewidth }
5782         }
5783     }
5784     \pgfusepath { clip }
5785     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5786     }
5787 }

```

The macro `\@@_actually_color:` will actually fill all the rectangles, color by color (using the sequence `\l_@@_colors_seq` and all the token lists of the form `\l_@@_color_i_tl`).

```

5788 \cs_new_protected:Npn \@@_actually_color:
5789 {
5790     \pgfpicture
5791     \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.

```

5792     \@@_clip_with_rounded_corners:
5793     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5794     {
5795         \int_compare:nNnTF { ##1 } = 1
5796         {
5797             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5798             \use:c { g_@@_color _ 1 _tl }
5799             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5800         }
5801         {
5802             \begin { pgfscope }
5803                 \@@_color_opacity: ##2
5804                 \use:c { g_@@_color _ ##1 _tl }
5805                 \tl_gclear:c { g_@@_color _ ##1 _tl }
5806                 \pgfusepath { fill }
5807             \end { pgfscope }
5808         }
5809     }
5810     \endpgfpicture
5811 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5812 \cs_new_protected:Npn \@@_color_opacity:
5813 {
5814     \peek_meaning:NTF [
5815         \@@_color_opacity:w
5816         { \@@_color_opacity:w [ ] }
5817     }

```

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.

```

5818 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]

```

```

5819 {
5820   \tl_clear:N \l_tmpa_tl
5821   \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.
5822   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5823   \tl_if_empty:NTF \l_tmpb_tl
5824     \@declaredcolor
5825     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5826 }

```

The following set of keys is used by the command `\@@_color_opacity:w`.

```

5827 \keys_define:nn { nicematrix / color-opacity }
5828 {
5829   opacity .tl_set:N          = \l_tmpa_tl ,
5830   opacity .value_required:n = true
5831 }

```

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

```

5832 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5833 {
5834   \def \l_@@_rows_tl { #1 }
5835   \def \l_@@_cols_tl { #2 }
5836   \@@_cartesian_path:
5837 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5838 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5839 {
5840   \tl_if_blank:nF { #2 }
5841   {
5842     \@@_add_to_colors_seq:en
5843     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5844     { \@@_cartesian_color:nn { #3 } { - } }
5845   }
5846 }

```

Here an example: `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5847 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5848 {
5849   \tl_if_blank:nF { #2 }
5850   {
5851     \@@_add_to_colors_seq:en
5852     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5853     { \@@_cartesian_color:nn { - } { #3 } }
5854   }
5855 }

```

Here is an example: `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5856 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5857 {
5858   \tl_if_blank:nF { #2 }
5859   {
5860     \@@_add_to_colors_seq:en
5861     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5862     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5863   }
5864 }

```


The last argument is the radius of the corners of the rectangle.

```

5865 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5866 {
5867   \tl_if_blank:nF { #2 }
5868   {
5869     \@@_add_to_colors_seq:en
5870     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5871     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5872   }
5873 }

```

The last argument is the radius of the corners of the rectangle.

```

5874 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5875 {
5876   \@@_cut_on_hyphen:w #1 \q_stop
5877   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5878   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5879   \@@_cut_on_hyphen:w #2 \q_stop
5880   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5881   \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`.

```

5882   \@@_cartesian_path:n { #3 }
5883 }

```

Here is an example: `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5884 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5885 {
5886   \clist_map_inline:nn { #3 }
5887   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5888 }

5889 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5890 {
5891   \int_step_inline:nn \c@iRow
5892   {
5893     \int_step_inline:nn \c@jCol
5894     {
5895       \int_if_even:nTF { #####1 + ##1 }
5896       { \@@_cellcolor [ #1 ] { #2 } }
5897       { \@@_cellcolor [ #1 ] { #3 } }
5898       { ##1 - #####1 }
5899     }
5900   }
5901 }

```

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”.

```

5902 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5903 {
5904   \@@_rectanglecolor [ #1 ] { #2 }
5905   { 1 - 1 }
5906   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5907 }

5908 \keys_define:nn { nicematrix / rowcolors }
5909 {
5910   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,

```

```

5911     respect-blocks .default:n = true ,
5912     cols .tl_set:N = \l_@@_cols_tl ,
5913     restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5914     restart .default:n = true ,
5915     unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5916 }

```

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*.

```

5917 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5918 {

```

`\l_@@_colors_seq` will be the list of colors.

```

5919     \pgfpicture
5920     \pgf@relevantforpicturesizefalse
5921     \seq_clear_new:N \l_@@_colors_seq
5922     \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5923     \tl_clear_new:N \l_@@_cols_tl
5924     \tl_set:Nn \l_@@_cols_tl { - }
5925     \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).

```

5926     \int_zero_new:N \l_@@_color_int
5927     \int_set:Nn \l_@@_color_int 1
5928     \bool_if:NT \l_@@_respect_blocks_bool
5929     {

```

We don't want to take into account a block which is entirely 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`).

```

5930         % modified 2026-02-18
5931         \seq_set_filter:Nnn \l_tmpa_seq \g_@@_pos_of_blocks_seq
5932         { \@@_not_in_exterior_p:nnnnn ##1 }
5933     }

```

#2 is the list of intervals of rows.

```

5934     \clist_map_inline:nn { #2 }
5935     {
5936         \tl_set:Nn \l_tmpa_tl { ##1 }
5937         \tl_if_in:NnTF \l_tmpa_tl { - }
5938         { \@@_cut_on_hyphen:w ##1 \q_stop }
5939         { \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.

```

5940         \int_set:Nn \l_tmpa_int \l_tmpa_tl
5941         \int_set:Nn \l_@@_color_int
5942         { \bool_if:NNTF \l_@@_rowcolors_restart_bool { 1 } \l_tmpa_tl }
5943         \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5944         \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5945         {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5946         \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).

```

5947         \bool_if:NT \l_@@_respect_blocks_bool
5948         {
5949             \seq_set_filter:Nnn \l_tmpb_seq \l_tmpa_seq
5950             { \@@_intersect_our_row_p:nnnnn ####1 }
5951             \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5952     }
5953     \tl_set:Nx \l_@@_rows_tl
5954     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
\l_@@_tmpc_tl will be the color that we will use.
5955     \tl_set:Nx \l_@@_color_tl
5956     {
5957         \@@_color_index:n
5958         {
5959             \int_mod:nn
5960             { \l_@@_color_int - 1 }
5961             { \seq_count:N \l_@@_colors_seq }
5962             + 1
5963         }
5964     }
5965     \tl_if_empty:NF \l_@@_color_tl
5966     {
5967         \@@_add_to_colors_seq:ee
5968         { \tl_if_blank:NF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5969         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5970     }
5971     \int_incr:N \l_@@_color_int
5972     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5973 }
5974 }
5975 \endpgfpicture
5976 }

```

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.

```

5977 \cs_new:Npn \@@_color_index:n #1
5978 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5979     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5980     { \@@_color_index:n { #1 - 1 } }
5981     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5982 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is a optional argument between square brackets is provided by currying.

```

5983 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5984 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5985 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5986 {
5987     \int_compare:nNnT { #3 } > \l_tmpb_int
5988     { \int_set:Nn \l_tmpb_int { #3 } }
5989 }

5990 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5991 {
5992     \int_if_zero:nTF { #4 }
5993     \prg_return_false:
5994     {
5995         \int_compare:nNnTF { #2 } > \c@jCol
5996         \prg_return_false:
5997         \prg_return_true:
5998     }
5999 }

```

The following command return true when the block intersects the row \l_tmpa_int.

```

6000 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
6001 {
6002   \int_compare:nNnTF { #1 } > \l_tmpa_int
6003     \prg_return_false:
6004     {
6005       \int_compare:nNnTF \l_tmpa_int > { #3 }
6006         \prg_return_false:
6007         \prg_return_true:
6008     }
6009 }

```

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).

```

6010 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
6011 {
6012   \dim_compare:nNnTF { #1 } = \c_zero_dim
6013   {
6014     \bool_if:NTF \l_@@_nocolor_used_bool
6015       { \@@_cartesian_path_normal_ii: }
6016       {
6017         \clist_if_empty:NTF \l_@@_corners_cells_clist
6018           { \@@_cartesian_path_normal_i:n { #1 } }
6019           { \@@_cartesian_path_normal_ii: }
6020       }
6021   }
6022   { \@@_cartesian_path_normal_i:n { #1 } }
6023 }

```

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.

```

6024 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
6025 {
6026   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

6027 \clist_map_inline:Nn \l_@@_cols_tl
6028 {

```

We use \def instead of \tl_set:Nn for efficiency only.

```

6029   \def \l_tmpa_tl { ##1 }
6030   \tl_if_in:NnTF \l_tmpa_tl { - }
6031     { \@@_cut_on_hyphen:w ##1 \q_stop }
6032     { \def \l_tmpb_tl { ##1 } }
6033   \tl_if_empty:NTF \l_tmpa_tl
6034     { \def \l_tmpa_tl { 1 } }
6035     {
6036       \str_if_eq:eeT \l_tmpa_tl { * }
6037       { \def \l_tmpa_tl { 1 } }
6038     }
6039   \int_compare:nNnT \l_tmpa_tl > \g_@@_col_total_int
6040     { \@@_error:n { Invalid~col~number } }
6041   \tl_if_empty:NTF \l_tmpb_tl
6042     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6043     {
6044       \str_if_eq:eeT \l_tmpb_tl { * }
6045       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6046     }

```

```

6047 \int_compare:nNt \l_tmpb_tl > \g_@@_col_total_int
6048 { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

6049 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6050 \@@_qpoint:n { col - \l_tmpa_tl }
6051 \int_compare:nNtF \l_@@_first_col_int = \l_tmpa_tl
6052 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6053 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6054 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
6055 \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.

```

6056 \clist_map_inline:Nn \l_@@_rows_tl
6057 {
6058   \def \l_tmpa_tl { #####1 }
6059   \tl_if_in:NnTF \l_tmpa_tl { - }
6060   { \@@_cut_on_hyphen:w #####1 \q_stop }
6061   { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
6062   \tl_if_empty:NtF \l_tmpa_tl
6063   { \def \l_tmpa_tl { 1 } }
6064   {
6065     \str_if_eq:eeT \l_tmpa_tl { * }
6066     { \def \l_tmpa_tl { 1 } }
6067   }
6068   \tl_if_empty:NtF \l_tmpb_tl
6069   { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6070   {
6071     \str_if_eq:eeT \l_tmpb_tl { * }
6072     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6073   }
6074   \int_compare:nNt \l_tmpa_tl > \g_@@_row_total_int
6075   { \@@_error:n { Invalid-row-number } }
6076   \int_compare:nNt \l_tmpb_tl > \g_@@_row_total_int
6077   { \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`.

```

6078 \cs_if_exist:cF
6079 { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
6080 {
6081   \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
6082   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6083   \@@_qpoint:n { row - \l_tmpa_tl }
6084   \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6085   \pgfpathrectanglecorners
6086   { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6087   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6088 }
6089 }
6090 }
6091 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

6092 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
6093 {
6094   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6095   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6096 \clist_map_inline:Nn \l_@@_cols_tl
6097 {
6098   \@@_qpoint:n { col - ##1 }
6099   \int_compare:nNtF \l_@@_first_col_int = { ##1 }
6100   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }

```

```

6101      { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6102      \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
6103      \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

6104      \clist_map_inline:Nn \l_@@_rows_tl
6105      {
6106          \@@_if_in_corner:nF { #####1 - ##1 }
6107          {
6108              \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
6109              \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6110              \@@_qpoint:n { row - #####1 }
6111              \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6112              \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
6113              {
6114                  \pgfpathrectanglecorners
6115                  { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6116                  { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6117              }
6118          }
6119      }
6120  }
6121 }

```

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`).

```

6122 \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

```

6123 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
6124 {
6125     \bool_set_true:N \l_@@_nocolor_used_bool
6126     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6127     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6128     \clist_map_inline:Nn \l_@@_rows_tl
6129     {
6130         \clist_map_inline:Nn \l_@@_cols_tl
6131         { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
6132     }
6133 }

```

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.

```

6134 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
6135 {
6136     \clist_set_eq:NN \l_tmpa_clist #1
6137     \clist_clear:N #1
6138     \clist_map_inline:Nn \l_tmpa_clist
6139     {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

6140     \def \l_tmpa_tl { ##1 }
6141     \tl_if_in:NnTF \l_tmpa_tl { - }
6142     { \@@_cut_on_hyphen:w ##1 \q_stop }
6143     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
6144     \bool_lazy_or:nnT
6145     { \str_if_eq_p:ee \l_tmpa_tl { * } }

```

```

6146     { \tl_if_blank_p:o \l_tmpa_tl }
6147     { \def \l_tmpa_tl { 1 } }
6148     \bool_lazy_or:nnT
6149     { \str_if_eq_p:ee \l_tmpb_tl { * } }
6150     { \tl_if_blank_p:o \l_tmpb_tl }
6151     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6152     \int_compare:nNnT \l_tmpb_tl > { #2 }
6153     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6154     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
6155     { \clist_put_right:Nn #1 { ####1 } }
6156   }
6157 }

```

The following command will be linked to `\cellcolor` in the tabular.

```

6158 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
6159 {
6160   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6161   {

```

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`).

```

6162     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
6163     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6164   }
6165   \ignorespaces
6166 }

6167 \NewDocumentCommand \@@_cellcolor_error { 0 { } m }
6168 { \@@_error:n { cellcolor~in~Block } }
6169 % \end{macrocode}
6170 %
6171 % \begin{macrocode}
6172 \NewDocumentCommand \@@_rowcolor_error { 0 { } m }
6173 { \@@_error:n { rowcolor~in~Block } }
6174 % \end{macrocode}
6175 %
6176 % \bigskip
6177 % The following command will be linked to |\rowcolor| in the tabular.
6178 % \begin{macrocode}
6179 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
6180 {
6181   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6182   {
6183     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
6184     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6185     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
6186   }
6187   \ignorespaces
6188 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by currying).

```

6189 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
6190 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around `#2` and `#3` are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

6191 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
6192 {

```

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`.

```

6193 \seq_gclear:N \g_tmpa_seq
6194 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6195   { \@@_rowlistcolors_tabular:nnnn ##1 }
6196 \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`.

```

6197 \seq_gput_right:Ne \g_@@_rowlistcolors_seq
6198   {
6199     { \int_use:N \c@iRow }
6200     { \exp_not:n { #1 } }
6201     { \exp_not:n { #2 } }
6202     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
6203   }
6204 \ignorespaces
6205 }

```

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`).

```

6206 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
6207   {
6208     \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).

```

6209   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
6210   {
6211     \tl_gput_right:Ne \g_@@_pre_code_before_tl
6212     {
6213       \@@_rowlistcolors
6214       [ \exp_not:n { #2 } ]
6215       { #1 - \int_eval:n { \c@iRow - 1 } }
6216       { \exp_not:n { #3 } }
6217       [ \exp_not:n { #4 } ]
6218     }
6219   }
6220 }

```

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.

```

6221 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
6222   {
6223     \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6224       { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
6225     \seq_gclear:N \g_@@_rowlistcolors_seq
6226   }

```



```

6227 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6228 {
6229   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6230   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6231 }

```

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.

```

6232 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
6233 {

```

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).

```

6234   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
6235   {

```

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).

```

6236     \tl_gput_left:Ne \g_@@_pre_code_before_tl
6237     {
6238       \exp_not:N \columncolor [ #1 ]
6239       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6240     }
6241   }
6242 }

```

```

6243 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6244 {
6245   \clist_map_inline:nn { #1 }
6246   {
6247     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6248     { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6249     \columncolor { nocolor } { ##1 }
6250   }
6251 }

```

```

6252 \cs_new_protected:Npn \@@_EmptyRow:n #1
6253 {
6254   \clist_map_inline:nn { #1 }
6255   {
6256     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6257     { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6258     \rowcolor { nocolor } { ##1 }
6259   }
6260 }

```

22 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnntype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`). That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```
6261 \cs_new_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`.

```
6262 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6263 {
6264   \int_if_zero:nTF \l_@@_first_col_int
6265   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6266   {
6267     \int_if_zero:nTF \c@jCol
6268     {
6269       \int_compare:nNnF \c@iRow = { -1 }
6270       {
6271         \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 }
6272         { #1 }
6273       }
6274     }
6275     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6276   }
6277 }
```

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.

```
6278 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6279 {
6280   \int_if_zero:nF \c@iRow
6281   {
6282     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6283     {
6284       \int_compare:nNnT \c@jCol > \c_zero_int
6285       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6286     }
6287   }
6288 }
```

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`.

```
6289 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6290 {
6291   \IfPackageLoadedTF { tikz }
6292   {
6293     \IfPackageLoadedTF { booktabs }
6294     { #2 }
6295     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6296   }
6297   { \@@_error:nn { TopRule~without~tikz } { #1 } }
6298 }

6299 \NewExpandableDocumentCommand { \@@_TopRule } { }
6300 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }

6301 \cs_new:Npn \@@_TopRule_i:
6302 {
6303   \noalign \bgroup
6304   \peek_meaning:NTF [
6305   { \@@_TopRule_ii: }
```

```

6306         { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6307     }
6308 \NewDocumentCommand \@@_TopRule_ii: { o }
6309 {
6310     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6311     {
6312         \@@_hline:n
6313         {
6314             position = \int_eval:n { \c@iRow + 1 } ,
6315             tikz =
6316             {
6317                 line~width = #1 ,
6318                 yshift = 0.25 \arrayrulewidth ,
6319                 shorten~< = - 0.5 \arrayrulewidth
6320             } ,
6321             total~width = #1
6322         }
6323     }
6324     \skip_vertical:n { \belowrulesep + #1 }
6325     \egroup
6326 }
6327 \NewExpandableDocumentCommand { \@@_BottomRule } { } { }
6328 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }
6329 \cs_new:Npn \@@_BottomRule_i:
6330 {
6331     \noalign \bgroup
6332     \peek_meaning:NTF [
6333         { \@@_BottomRule_ii: }
6334         { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6335     }
6336 \NewDocumentCommand \@@_BottomRule_ii: { o }
6337 {
6338     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6339     {
6340         \@@_hline:n
6341         {
6342             position = \int_eval:n { \c@iRow + 1 } ,
6343             tikz =
6344             {
6345                 line~width = #1 ,
6346                 yshift = 0.25 \arrayrulewidth ,
6347                 shorten~< = - 0.5 \arrayrulewidth
6348             } ,
6349             total~width = #1 ,
6350         }
6351     }
6352     \skip_vertical:N \aboverulesep
6353     \@@_create_row_node_i:
6354     \skip_vertical:n { #1 }
6355     \egroup
6356 }
6357 \NewExpandableDocumentCommand { \@@_MidRule } { } { }
6358 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6359 \cs_new:Npn \@@_MidRule_i:
6360 {
6361     \noalign \bgroup
6362     \peek_meaning:NTF [
6363         { \@@_MidRule_ii: }
6364         { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6365     }
6366 \NewDocumentCommand \@@_MidRule_ii: { o }

```

```

6367 {
6368   \skip_vertical:N \aboverulesep
6369   \@@_create_row_node_i:
6370   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6371   {
6372     \@@_hline:n
6373     {
6374       position = \int_eval:n { \c@iRow + 1 } ,
6375       tikz =
6376       {
6377         line-width = #1 ,
6378         yshift = 0.25 \arrayrulewidth ,
6379         shorten~< = - 0.5 \arrayrulewidth
6380       } ,
6381       total-width = #1 ,
6382     }
6383   }
6384   \skip_vertical:n { \belowrulesep + #1 }
6385   \egroup
6386 }

```

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.

```

6387 \keys_define:nn { nicematrix / Rules }
6388 {
6389   position .int_set:N = \l_@@_position_int ,
6390   position .value_required:n = true ,
6391   start .int_set:N = \l_@@_start_int ,
6392   end .code:n =
6393     \bool_lazy_or:nnTF
6394     { \tl_if_empty_p:n { #1 } }
6395     { \str_if_eq_p:ee { #1 } { last } }
6396     { \int_set_eq:NN \l_@@_end_int \c@jCol }
6397     { \int_set:Nn \l_@@_end_int { #1 } }
6398 }

```

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.

```

6399 \keys_define:nn { nicematrix / RulesBis }
6400 {
6401   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6402   multiplicity .initial:n = 1 ,
6403   dotted .bool_set:N = \l_@@_dotted_bool ,
6404   dotted .initial:n = false ,
6405   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]`.

```

6406   color .code:n =
6407     \@@_set_CTarc:n { #1 }

```

```

6408     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6409     color .value_required:n = true ,
6410     sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6411     sep-color .value_required:n = true ,
6412     total-width .dim_set:N = \l_@@_rule_width_dim ,
6413     total-width .value_required:n = true ,
6414     width .meta:n = { total-width = #1 } ,
6415     unknown .code:n =
6416         \@@_unknown_key:nn
6417         { nicematrix / RulesBis }
6418         { Unknown-key-for-RulesBis } ,
6419     tikz .value_required:n = true
6420 }

```

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.

```

6421 \hook_gput_code:nnn { begindocument } { . }
6422 {
6423     \IfPackageLoadedTF { tikz }
6424     {
6425         \keys_define:nn { nicematrix / RulesBis }
6426         { tikz .code:n = \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6427     }
6428     {
6429         \keys_define:nn { nicematrix / RulesBis }
6430         { tikz .code:n = \@@_error:n { tikz-without-tikz } }
6431     }
6432 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6433 \cs_new_protected:Npn \@@_vline:n #1
6434 {

```

The group is for the options.

```

6435     \group_begin:
6436     \int_set_eq:NN \l_@@_end_int \c@iRow
6437     \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).

```

6438     \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6439     \@@_vline_i:
6440     \group_end:
6441 }
6442 \cs_new_protected:Npn \@@_vline_i:
6443 {
6444     \tl_clear:N \l_@@_tikz_rule_tl
6445     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl

```

`\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`.

```

6446     \tl_set:Nn \l_tmpb_tl { \int_use:N \l_@@_position_int }
6447     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6448     \l_tmpa_tl
6449     {

```

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.

```

6450     \bool_gset_true:N \g_tmpa_bool

```

```

6451 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6452 { \@@_test_vline_in_block:nnnnn ##1 }
6453 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6454 { \@@_test_vline_in_block:nnnnn ##1 }
6455 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6456 { \@@_test_vline_in_stroken_block:nnnn ##1 }
6457 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6458 \bool_if:NTF \g_tmpa_bool
6459 {
6460 \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.

```

6461 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6462 }
6463 {
6464 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6465 {
6466 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6467 \@@_vline_ii:
6468 \int_zero:N \l_@@_local_start_int
6469 }
6470 }
6471 }
6472 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6473 {
6474 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6475 \@@_vline_ii:
6476 }
6477 }

6478 \cs_new_protected:Npn \@@_test_in_corner_v:
6479 {
6480 \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6481 {
6482 \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6483 { \bool_set_false:N \g_tmpa_bool }
6484 }
6485 {
6486 \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6487 {
6488 \int_compare:nNnTF \l_tmpb_tl = 1
6489 { \bool_set_false:N \g_tmpa_bool }
6490 {
6491 \@@_if_in_corner:nT
6492 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6493 { \bool_set_false:N \g_tmpa_bool }
6494 }
6495 }
6496 }
6497 }

6498 \cs_new_protected:Npn \@@_vline_ii:
6499 {
6500 % \tl_clear:N \l_@@_tikz_rule_tl
6501 % \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6502 \bool_if:NTF \l_@@_dotted_bool
6503 \@@_vline_iv:
6504 {
6505 \tl_if_empty:NTF \l_@@_tikz_rule_tl
6506 \@@_vline_iii:
6507 \@@_vline_v:

```

```

6508 }
6509 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

6510 \cs_new_protected:Npn \@@_vline_iii:
6511 {
6512   \pgfpicture
6513   \pgfrememberpicturepositiononpagetrue
6514   \pgf@relevantforpicturesizefalse
6515   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6516   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6517   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6518   \dim_set:Nn \l_tmpb_dim
6519   {
6520     \pgf@x
6521     - 0.5 \l_@@_rule_width_dim
6522     +
6523     ( \arrayrulewidth * \l_@@_multiplicity_int
6524       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6525   }
6526   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6527   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6528   \bool_lazy_all:nT
6529   {
6530     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
6531     { \cs_if_exist_p:N \CT@drsc@ }
6532     { ! \tl_if_blank_p:o \CT@drsc@ }
6533   }
6534   {
6535     \group_begin:
6536     \CT@drsc@
6537     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6538     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6539     \dim_set:Nn \l_@@_tmpd_dim
6540     {
6541       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6542       * ( \l_@@_multiplicity_int - 1 )
6543     }
6544     \pgfpathrectanglecorners
6545     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6546     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6547     \pgfusepath { fill }
6548     \group_end:
6549   }
6550   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6551   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6552   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6553   {
6554     \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6555     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6556     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6557   }
6558   \CT@arc@
6559   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6560   \pgfsetrectcap
6561   \pgfusepathqstroke
6562   \endpgfpicture
6563 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6564 \cs_new_protected:Npn \@@_vline_iv:
6565 {

```

```

6566 \pgfpicture
6567 \pgfrememberpicturepositiononpagetrue
6568 \pgf@relevantforpicturesizefalse
6569 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6570 \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6571 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6572 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6573 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6574 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6575 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6576 \CT@arc@
6577 \@@_draw_line:
6578 \endpgfpicture
6579 }

```

The following code is for the case when the user uses the key `tikz`.

```

6580 \cs_new_protected:Npn \@@_vline_v:
6581 {
6582   \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).

```

6583   \CT@arc@
6584   \tl_if_empty:NF \l_@@_rule_color_tl
6585   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6586   \pgfrememberpicturepositiononpagetrue
6587   \pgf@relevantforpicturesizefalse
6588   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6589   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6590   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6591   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6592   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6593   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6594   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6595   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6596   ( \l_tmpb_dim , \l_tmpa_dim ) --
6597   ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6598   \end { tikzpicture }
6599 }

```

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).

```

6600 \cs_new_protected:Npn \@@_draw_vlines:
6601 {
6602   \int_step_inline:nnn
6603   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6604   {
6605     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6606     \c@jCol
6607     { \int_eval:n { \c@jCol + 1 } } }
6608   }
6609   {
6610     \str_if_eq:eeF \l_@@_vlines_clist { all }
6611     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6612     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6613   }
6614 }

```


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}`.

```
6615 \cs_new_protected:Npn \@@_hline:n #1
6616 {
```

The group is for the options.

```
6617   \group_begin:
6618   \int_set_eq:NN \l_@@_end_int \c@jCol
6619   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6620   \@@_hline_i:
6621   \group_end:
6622 }
```

```
6623 \cs_new_protected:Npn \@@_hline_i:
6624 {
6625   \tl_clear:N \l_@@_tikz_rule_tl
6626   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
```

`\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`.

```
6627   \tl_set:N \l_tmpa_tl { \int_use:N \l_@@_position_int }
6628   \int_step_variable:nnN \l_@@_start_int \l_@@_end_int
6629   \l_tmpb_tl
6630   {
```

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.

```
6631       \bool_gset_true:N \g_tmpa_bool
```

We test whether we are in a block.

```
6632   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6633   { \@@_test_hline_in_block:nnnnn ##1 }
6634   \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6635   { \@@_test_hline_in_block:nnnnn ##1 }
6636   \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6637   { \@@_test_hline_in_stroken_block:nnnn ##1 }
6638   \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_h: }
6639   \bool_if:NTF \g_tmpa_bool
6640   {
6641     \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.

```
6642       { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6643     }
6644     {
6645       \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6646       {
6647         \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6648         \@@_hline_ii:
6649         \int_zero:N \l_@@_local_start_int
6650       }
6651     }
6652   }
6653   \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6654   {
6655     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6656     \@@_hline_ii:
6657   }
6658 }
```

```

6659 \cs_new_protected:Npn \@@_test_in_corner_h:
6660 {
6661   \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6662   {
6663     \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6664     { \bool_set_false:N \g_tmpa_bool }
6665   }
6666   {
6667     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6668     {
6669       \int_compare:nNnTF \l_tmpa_tl = 1
6670       { \bool_set_false:N \g_tmpa_bool }
6671       {
6672         \@@_if_in_corner:nT
6673         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6674         { \bool_set_false:N \g_tmpa_bool }
6675       }
6676     }
6677   }
6678 }

6679 \cs_new_protected:Npn \@@_hline_ii:
6680 {
6681   % \tl_clear:N \l_@@_tikz_rule_tl
6682   % \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6683   \bool_if:NTF \l_@@_dotted_bool
6684   \@@_hline_iv:
6685   {
6686     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6687     \@@_hline_iii:
6688     \@@_hline_v:
6689   }
6690 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6691 \cs_new_protected:Npn \@@_hline_iii:
6692 {
6693   \pgfpicture
6694   \pgfrememberpicturepositiononpagetrue
6695   \pgf@relevantforpicturesizefalse
6696   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6697   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6698   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6699   \dim_set:Nn \l_tmpb_dim
6700   {
6701     \pgf@y
6702     - 0.5 \l_@@_rule_width_dim
6703     +
6704     ( \arrayrulewidth * \l_@@_multiplicity_int
6705       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6706   }
6707   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6708   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6709   \bool_lazy_all:nT
6710   {
6711     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
6712     { \cs_if_exist_p:N \CT@drsc@ }
6713     { ! \tl_if_blank_p:o \CT@drsc@ }
6714   }
6715   {
6716     \group_begin:
6717     \CT@drsc@

```

```

6718 \dim_set:Nn \l_@@_tmpd_dim
6719 {
6720 \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6721 * ( \l_@@_multiplicity_int - 1 )
6722 }
6723 \pgfpathrectanglecorners
6724 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6725 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6726 \pgfusepathqfill
6727 \group_end:
6728 }
6729 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6730 \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6731 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6732 {
6733 \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6734 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6735 \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6736 }
6737 \CT@arc@
6738 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6739 \pgfsetrectcap
6740 \pgfusepathqstroke
6741 \endpgfpicture
6742 }

```

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}

```

$$\left[\begin{array}{cccc} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{array} \right]$$

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}

```

$$\left[\begin{array}{cccc} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{array} \right]$$

```

6743 \cs_new_protected:Npn \@@_hline_iv:
6744 {
6745 \pgfpicture
6746 \pgfrememberpicturepositiononpagetrue
6747 \pgf@relevantforpicturesizefalse
6748 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6749 \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6750 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6751 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6752 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6753 \int_compare:nNnT \l_@@_local_start_int = 1
6754 {
6755 \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6756 \bool_if:NF \g_@@_delims_bool
6757 { \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.

```

6758 \tl_if_eq:NnF \g_@@_left_delim_tl (

```

```

6759         { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6760     }
6761     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6762     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6763     \int_compare:nNnT \l_@@_local_end_int = \c@jCol
6764     {
6765         \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6766         \bool_if:NF \g_@@_delims_bool
6767         { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6768         \tl_if_eq:NnF \g_@@_right_delim_tl )
6769         { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6770     }
6771     \CT@arc@
6772     \@@_draw_line:
6773     \endpgfpicture
6774 }

```

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`).

```

6775 \cs_new_protected:Npn \@@_hline_v:
6776 {
6777     \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).

```

6778     \CT@arc@
6779     \tl_if_empty:NF \l_@@_rule_color_tl
6780     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6781     \pgfrememberpicturepositiononpagetrue
6782     \pgf@relevantforpicturesizefalse
6783     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6784     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6785     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6786     \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6787     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6788     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6789     \exp_args:No \tikzset \l_@@_tikz_rule_tl
6790     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6791     ( \l_tmpa_dim , \l_tmpb_dim ) --
6792     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6793     \end { tikzpicture }
6794 }

```

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).

```

6795 \cs_new_protected:Npn \@@_draw_hlines:
6796 {
6797     \int_step_inline:nnn
6798     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6799     {
6800         \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6801         \c@iRow
6802         { \int_eval:n { \c@iRow + 1 } } }
6803     }
6804     {
6805         \str_if_eq:eeF \l_@@_hlines_clist { all }
6806         { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6807         { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6808     }
6809 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```
6810 \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.

```
6811 \cs_set:Npn \@@_Hline_i:n #1
6812 {
6813   \peek_remove_spaces:n
6814   {
6815     \peek_meaning:NTF \Hline
6816     { \@@_Hline_ii:nn { #1 + 1 } }
6817     { \@@_Hline_iii:n { #1 } }
6818   }
6819 }

6820 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }

6821 \cs_set:Npn \@@_Hline_iii:n #1
6822 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }

6823 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6824 {
6825   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6826   \skip_vertical:N \l_@@_rule_width_dim
6827   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6828   {
6829     \@@_hline:n
6830     {
6831       multiplicity = #1 ,
6832       position = \int_eval:n { \c@iRow + 1 } ,
6833       total-width = \dim_use:N \l_@@_rule_width_dim ,
6834       #2
6835     }
6836   }
6837   \egroup
6838 }
```

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`).

```
6839 \cs_new_protected:Npn \@@_custom_line:n #1
6840 {
6841   \str_clear_new:N \l_@@_command_str
6842   \str_clear_new:N \l_@@_ccommand_str
6843   \str_clear_new:N \l_@@_letter_str
6844   \tl_clear_new:N \l_@@_other_keys_tl
6845   \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
6846   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
6847   {
6848     \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }
6849   }
```

We delete the last character which is a space.

```
6849   \str_set:Ne \l_@@_command_str
6850   { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
6851 }
6852 \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
6853 {
6854   \str_set:Ne \l_@@_ccommand_str
6855   { \str_tail:N \l_@@_ccommand_str }
6856   \str_set:Ne \l_@@_ccommand_str
6857   { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
6858 }
```

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.

```

6859 \bool_lazy_all:nTF
6860 {
6861   { \str_if_empty_p:N \l_@@_letter_str }
6862   { \str_if_empty_p:N \l_@@_command_str }
6863   { \str_if_empty_p:N \l_@@_ccommand_str }
6864 }
6865 { \@@_error:n { No~letter~and~no~command } }
6866 { \@@_custom_line_i:o \l_@@_other_keys_tl }
6867 }

6868 \keys_define:nn { nicematrix / custom-line }
6869 {
6870   letter .str_set:N = \l_@@_letter_str ,
6871   letter .value_required:n = true ,
6872   command .str_set:N = \l_@@_command_str ,
6873   command .value_required:n = true ,
6874   ccommand .str_set:N = \l_@@_ccommand_str ,
6875   ccommand .value_required:n = true ,
6876 }

```

```

6877 \cs_new_protected:Npn \@@_custom_line_i:n #1
6878 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6879 \bool_set_false:N \l_@@_tikz_rule_bool
6880 \bool_set_false:N \l_@@_dotted_rule_bool
6881 \bool_set_false:N \l_@@_color_bool

6882 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6883 \bool_if:NT \l_@@_tikz_rule_bool
6884 {
6885   \IfPackageLoadedF { tikz }
6886   { \@@_error:n { tikz~in~custom~line~without~tikz } }
6887   \bool_if:NT \l_@@_color_bool
6888   { \@@_error:n { color~in~custom~line~with~tikz } }
6889 }
6890 \bool_if:NT \l_@@_dotted_rule_bool
6891 {
6892   \int_compare:nNnT \l_@@_multiplicity_int > 1
6893   { \@@_error:n { key~multiplicity~with~dotted } }
6894 }
6895 \str_if_empty:NF \l_@@_letter_str
6896 {
6897   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6898   { \@@_error:n { Several~letters } }
6899   {
6900     \tl_if_in:NoTF
6901     \c_@@_forbidden_letters_str
6902     \l_@@_letter_str
6903     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6904   }

```

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.

```

6905 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6906 { \@@_v_custom_line:nn { #1 } }
6907 }

```

```

6908     }
6909   }
6910   \str_if_empty:NF \l_@@_command_str { \l_@@_h_custom_line:n { #1 } }
6911   \str_if_empty:NF \l_@@_ccommand_str { \l_@@_c_custom_line:n { #1 } }
6912 }
6913 \cs_generate_variant:Nn \l_@@_custom_line_i:n { o }
6914 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6915 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\l_@@_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.

```

6916 \keys_define:nn { nicematrix / custom-line-bis }
6917 {
6918   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6919   multiplicity .initial:n = 1 ,
6920   multiplicity .value_required:n = true ,
6921   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6922   color .value_required:n = true ,
6923   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6924   tikz .value_required:n = true ,
6925   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6926   dotted .value_forbidden:n = true ,
6927   total-width .code:n = { } ,
6928   total-width .value_required:n = true ,
6929   width .code:n = { } ,
6930   width .value_required:n = true ,
6931   sep-color .code:n = { } ,
6932   sep-color .value_required:n = true ,
6933   unknown .code:n =
6934     \l_@@_unknown_key:nn
6935     { nicematrix / custom-line-bis }
6936     { Unknown-key-for~custom-line }
6937 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6938 \bool_new:N \l_@@_dotted_rule_bool
6939 \bool_new:N \l_@@_tikz_rule_bool
6940 \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`.

```

6941 \keys_define:nn { nicematrix / custom-line-width }
6942 {
6943   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6944   multiplicity .initial:n = 1 ,
6945   multiplicity .value_required:n = true ,
6946   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6947   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6948     \bool_set_true:N \l_@@_total_width_bool ,
6949   total-width .value_required:n = true ,
6950   width .meta:n = { total-width = #1 } ,
6951   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6952 }

```

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 `\l_@@_hline:n` (which is in the internal `\CodeAfter`).

```

6953 \cs_new_protected:Npn \l_@@_h_custom_line:n #1
6954 {

```

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`).

```

6955 \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6956 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6957 }

```

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`).

```

6958 \cs_new_protected:Npn \@@_c_custom_line:n #1
6959 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

6960 \exp_args:Nc \DeclareExpandableDocumentCommand
6961 { nicematrix - \l_@@_ccommand_str }
6962 { 0 { } m }
6963 {
6964 \noalign
6965 {
6966 \@@_compute_rule_width:n { #1 , ##1 }
6967 \skip_vertical:n { \l_@@_rule_width_dim }
6968 \clist_map_inline:nn
6969 { ##2 }
6970 { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6971 }
6972 }
6973 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6974 }

```

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*.

```

6975 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6976 {
6977 \tl_if_in:nnTF { #2 } { - }
6978 { \@@_cut_on_hyphen:w #2 \q_stop }
6979 { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6980 \tl_gput_right:Ne \g_@@_pre_code_after_tl
6981 {
6982 \@@_hline:n
6983 {
6984 #1 ,
6985 start = \l_tmpa_tl ,
6986 end = \l_tmpb_tl ,
6987 position = \int_eval:n { \c@iRow + 1 } ,
6988 total-width = \dim_use:N \l_@@_rule_width_dim
6989 }
6990 }
6991 }

6992 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6993 {
6994 \bool_set_false:N \l_@@_tikz_rule_bool
6995 \bool_set_false:N \l_@@_total_width_bool
6996 \bool_set_false:N \l_@@_dotted_rule_bool
6997 \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6998 \bool_if:NF \l_@@_total_width_bool
6999 {
7000 \bool_if:NTF \l_@@_dotted_rule_bool
7001 { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
7002 {
7003 \bool_if:NF \l_@@_tikz_rule_bool
7004 {

```



```

7005         \dim_set:Nn \l_@@_rule_width_dim
7006         {
7007             \arrayrulewidth * \l_@@_multiplicity_int
7008             + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
7009         }
7010     }
7011 }
7012 }
7013 }

```

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

```

7014 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
7015 {
7016     \str_if_eq:nnTF { #2 } { [ ] }
7017     { \@@_v_custom_line_i:nw { #1 } [ ] }
7018     { \@@_v_custom_line_ii:nn { #2 } { #1 } }
7019 }
7020 \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
7021 { \@@_v_custom_line:nn { #1 , #2 } }
7022 \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
7023 {
7024     \@@_compute_rule_width:n { #2 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

7025     \tl_gput_right:Ne \g_@@_array_preamble_tl
7026     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
7027     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7028     {
7029         \@@_vline:n
7030         {
7031             #2 ,
7032             position = \int_eval:n { \c@jCol + 1 } ,
7033             total-width = \dim_use:N \l_@@_rule_width_dim
7034         }
7035     }
7036     \@@_rec_preamble:n #1
7037 }
7038 \@@_custom_line:n
7039 { letter = : , command = \hdottedline , ccommand = \cdottedline, dotted }
7040 \hook_gput_code:nnn { begindocument } { . }
7041 {
7042     \IfPackageLoadedT { tikz }
7043     {
7044         \cs_if_exist:cTF { tikz@library@decorations@loaded }
7045         {
7046             \@@_custom_line:n
7047             {
7048                 letter = ; ,
7049                 command = \hdashedline ,
7050                 ccommand = \cdashedline ,
7051                 tikz = { dash-pattern = on~4~pt~off~2pt, dash-expand-off } ,
7052                 total-width = \pgflinewidth
7053             }
7054         }
7055         {
7056             \@@_custom_line:n
7057             {
7058                 letter = ; ,
7059                 command = \hdashedline ,
7060                 ccommand = \cdashedline ,
7061                 tikz = { dash-pattern = on~4~pt~off~2pt } ,
7062                 total-width = \pgflinewidth

```

```

7063     }
7064   }
7065 }
7066 }

```

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.

```

7067 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
7068 {
7069   \int_compare:nNtT \l_tmpa_tl > { #1 }
7070   {
7071     \int_compare:nNtT \l_tmpa_tl < { #3 + 1 }
7072     {
7073       \int_compare:nNtT \l_tmpb_tl > { #2 - 1 }
7074       {
7075         \int_compare:nNtT \l_tmpb_tl < { #4 + 1 }
7076         { \bool_gset_false:N \g_tmpa_bool }
7077       }
7078     }
7079   }
7080 }

```

The same for vertical rules.

```

7081 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
7082 {
7083   \int_compare:nNtT \l_tmpa_tl > { #1 - 1 }
7084   {
7085     \int_compare:nNtT \l_tmpa_tl < { #3 + 1 }
7086     {
7087       \int_compare:nNtT \l_tmpb_tl > { #2 }
7088       {
7089         \int_compare:nNtT \l_tmpb_tl < { #4 + 1 }
7090         { \bool_gset_false:N \g_tmpa_bool }
7091       }
7092     }
7093   }
7094 }
7095 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
7096 {
7097   \int_compare:nNtT \l_tmpb_tl > { #2 - 1 }
7098   {
7099     \int_compare:nNtT \l_tmpb_tl < { #4 + 1 }
7100     {
7101       \int_compare:nNtTF \l_tmpa_tl = { #1 }
7102       { \bool_gset_false:N \g_tmpa_bool }
7103       {
7104         \int_compare:nNtT \l_tmpa_tl = { #3 + 1 }
7105         { \bool_gset_false:N \g_tmpa_bool }
7106       }
7107     }
7108   }
7109 }
7110 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
7111 {
7112   \int_compare:nNtT \l_tmpa_tl > { #1 - 1 }
7113   {
7114     \int_compare:nNtT \l_tmpa_tl < { #3 + 1 }
7115     {
7116       \int_compare:nNtTF \l_tmpb_tl = { #2 }

```

```

7117         { \bool_gset_false:N \g_tmpa_bool }
7118         {
7119             \int_compare:nNt \l_tmpb_tl = { #4 + 1 }
7120             { \bool_gset_false:N \g_tmpa_bool }
7121         }
7122     }
7123 }
7124 }

```

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.

```

7125 \cs_new_protected:Npn \@@_compute_corners:
7126 {
7127     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7128     { \@@_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`).

```

7129     \clist_clear:N \l_@@_corners_cells_clist
7130     \clist_map_inline:Nn \l_@@_corners_cells_clist
7131     {
7132         \str_case:nnF { ##1 }
7133         {
7134             { NW }
7135             { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
7136             { NE }
7137             { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
7138             { SW }
7139             { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
7140             { SE }
7141             { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
7142         }
7143         { \@@_error:nn { bad~corner } { ##1 } }
7144     }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

7145     \clist_if_empty:NF \l_@@_corners_cells_clist
7146     {

```

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.

```

7147         \tl_gput_right:Ne \g_@@_aux_tl
7148         {
7149             \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
7150             { \l_@@_corners_cells_clist }
7151         }
7152     }
7153 }

```

```

7154 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
7155 {
7156     \int_step_inline:nnn { #1 } { #3 }
7157     {

```

```

7158     \int_step_inline:nnn { #2 } { #4 }
7159     { \cs_set_nopar:cpn { @@ _ block _ ##1 - ###1 } { } }
7160   }
7161 }

7162 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
7163 {
7164   \cs_if_exist:cTF
7165     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
7166     \prg_return_true:
7167     \prg_return_false:
7168 }

```

“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.

```

7169 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
7170 {

```

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.

```

7171   \bool_set_false:N \l_tmpa_bool
7172   \int_zero_new:N \l_@@_last_empty_row_int
7173   \int_set:Nn \l_@@_last_empty_row_int { #1 }
7174   \int_step_inline:nnnn { #1 } { #3 } { #5 }
7175   {
7176     \bool_lazy_or:nnTF
7177     {
7178       \cs_if_exist_p:c
7179       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
7180     }
7181     { \@@_if_in_block_p:nn { ##1 } { #2 } }
7182     { \bool_set_true:N \l_tmpa_bool }
7183     {
7184       \bool_if:NF \l_tmpa_bool
7185       { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
7186     }
7187   }

```

Now, you determine the last empty cell in the row of number 1.

```

7188   \bool_set_false:N \l_tmpa_bool
7189   \int_zero_new:N \l_@@_last_empty_column_int
7190   \int_set:Nn \l_@@_last_empty_column_int { #2 }
7191   \int_step_inline:nnnn { #2 } { #4 } { #6 }
7192   {
7193     \bool_lazy_or:nnTF
7194     {
7195       \cs_if_exist_p:c
7196       { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
7197     }
7198     { \@@_if_in_block_p:nn { #1 } { ##1 } }

```

```

7199         { \bool_set_true:N \l_tmpa_bool }
7200     {
7201         \bool_if:NF \l_tmpa_bool
7202         { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
7203     }
7204 }

```

Now, we loop over the rows.

```

7205     \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
7206     {

```

We treat the row number ##1 with another loop.

```

7207         \bool_set_false:N \l_tmpa_bool
7208         \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
7209         {
7210             \bool_lazy_or:nnTF
7211             { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7212             { \@@_if_in_block_p:nn { ##1 } { #####1 } }
7213             { \bool_set_true:N \l_tmpa_bool }
7214         {
7215             \bool_if:NF \l_tmpa_bool
7216             {
7217                 \int_set:Nn \l_@@_last_empty_column_int { #####1 }
7218                 \clist_put_right:Nn
7219                 \l_@@_corners_cells_clist
7220                 { ##1 - #####1 }
7221                 \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7222             }
7223         }
7224     }
7225 }
7226 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

7227 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
7228 \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.

```

7229 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

7230 \keys_define:nn { nicematrix / NiceMatrixBlock }
7231 {
7232     auto-columns-width .code:n =
7233     {
7234         \bool_set_true:N \l_@@_block_auto_columns_width_bool
7235         \dim_gzero_new:N \g_@@_max_cell_width_dim
7236         \bool_set_true:N \l_@@_auto_columns_width_bool
7237     }
7238 }

```

```

7239 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
7240 {
7241   \int_gincr:N \g_@@_NiceMatrixBlock_int
7242   \dim_zero:N \l_@@_columns_width_dim
7243   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
7244   \bool_if:NT \l_@@_block_auto_columns_width_bool
7245   {
7246     \cs_if_exist:cT
7247     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
7248     {
7249       \dim_set:Nn \l_@@_columns_width_dim
7250       {
7251         \use:c
7252         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7253       }
7254     }
7255   }
7256 }

```

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`).

```

7257 {
7258   \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.

```

7259   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
7260   {
7261     \bool_if:NT \l_@@_block_auto_columns_width_bool
7262     {
7263       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
7264       \iow_shipout:Ne \@mainaux
7265       {
7266         \cs_gset:cpn
7267         { @@ _ 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.

```

7268         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7269       }
7270       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
7271     }
7272   }
7273   \ignorespacesafterend
7274 }

```

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).

```

7275 \cs_new_protected:Npn \@@_create_extra_nodes:
7276 {
7277   \bool_if:nTF \l_@@_medium_nodes_bool
7278   {
7279     \bool_if:NTF \l_@@_no_cell_nodes_bool
7280     { \@@_error:n { extra-nodes~with~no~cell~nodes } }
7281     {
7282       \bool_if:NTF \l_@@_large_nodes_bool

```

```

7283         \@@_create_medium_and_large_nodes:
7284         \@@_create_medium_nodes:
7285     }
7286 }
7287 {
7288     \bool_if:NT \l_@@_large_nodes_bool
7289     {
7290         \bool_if:NTF \l_@@_no_cell_nodes_bool
7291         { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7292         \@@_create_large_nodes:
7293     }
7294 }
7295 }

```

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`.

```

7296 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7297 {
7298     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7299     {
7300         \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7301         \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7302         \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7303         \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7304     }
7305     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7306     {
7307         \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7308         \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7309         \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7310         \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7311     }

```

We begin the two nested loops over the rows and the columns of the array.

```

7312     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7313     {
7314         \int_step_variable:nnNn
7315         \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.

```

7316     {
7317         \cs_if_exist:cT
7318         { 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`.

```

7319     {
7320         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7321         \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7322         { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7323         \seq_if_in:Nef \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7324         {
7325             \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7326             { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7327         }

```

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`.

```

7328         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7329         \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7330         { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
7331         \seq_if_in:Nef \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7332         {
7333             \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7334             { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
7335         }
7336     }
7337 }
7338 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7339 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7340 {
7341     \dim_compare:nNnT
7342     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7343     {
7344         \@@_qpoint:n { row - \@@_i: - base }
7345         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7346         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7347     }
7348 }
7349 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7350 {
7351     \dim_compare:nNnT
7352     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7353     {
7354         \@@_qpoint:n { col - \@@_j: }
7355         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7356         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7357     }
7358 }
7359 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7360 \cs_new_protected:Npn \@@_create_medium_nodes:
7361 {
7362     \pgfpicture
7363     \pgfrememberpicturepositiononpagetrue
7364     \pgf@relevantforpicturesizefalse
7365     \@@_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”.

```

7366     \tl_set:Nn \l_@@_suffix_tl { -medium }
7367     \@@_create_nodes:

```



```

7368 \endpgfpicture
7369 }

```

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:`.

```

7370 \cs_new_protected:Npn \@@_create_large_nodes:
7371 {
7372   \pgfpicture
7373   \pgfrememberpicturepositiononpagetrue
7374   \pgf@relevantforpicturesizefalse
7375   \@@_computations_for_medium_nodes:
7376   \@@_computations_for_large_nodes:
7377   \tl_set:Nn \l_@@_suffix_tl { - large }
7378   \@@_create_nodes:
7379   \endpgfpicture
7380 }
7381 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7382 {
7383   \pgfpicture
7384   \pgfrememberpicturepositiononpagetrue
7385   \pgf@relevantforpicturesizefalse
7386   \@@_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”.

```

7387   \tl_set:Nn \l_@@_suffix_tl { - medium }
7388   \@@_create_nodes:
7389   \@@_computations_for_large_nodes:
7390   \tl_set:Nn \l_@@_suffix_tl { - large }
7391   \@@_create_nodes:
7392   \endpgfpicture
7393 }

```

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.

```

7394 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7395 {
7396   \int_set:Nn \l_@@_first_row_int 1
7397   \int_set:Nn \l_@@_first_col_int 1

```

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`.

```

7398   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7399   {
7400     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7401     {
7402       (
7403         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7404         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7405       )
7406       / 2
7407     }
7408     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7409     { l_@@_row _ \@@_i: _ min _ dim }
7410   }
7411   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:

```

¹⁵If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7412 {
7413   \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7414   {
7415     (
7416       \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7417       \dim_use:c
7418         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7419     )
7420     / 2
7421   }
7422   \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7423   { l_@@_column _ \@@_j: _ max _ dim }
7424 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7425 \dim_sub:cn
7426   { l_@@_column _ 1 _ min _ dim }
7427   \l_@@_left_margin_dim
7428 \dim_add:cn
7429   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7430   \l_@@_right_margin_dim
7431 }

```

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).

```

7432 \cs_new_protected:Npn \@@_create_nodes:
7433 {
7434   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7435   {
7436     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7437     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7438       \@@_pgf_rect_node:nnnnn
7439       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7440       { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7441       { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7442       { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7443       { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7444       \str_if_empty:NF \l_@@_name_str
7445       {
7446         \pgfnodealias
7447         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7448         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7449       }
7450     }
7451   }
7452   \int_step_inline:nn \c@iRow
7453   {
7454     \pgfnodealias
7455     { \@@_env: - ##1 - last \l_@@_suffix_tl }
7456     { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7457   }
7458   \int_step_inline:nn \c@jCol
7459   {
7460     \pgfnodealias
7461     { \@@_env: - last - ##1 \l_@@_suffix_tl }
7462     { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7463   }

```

```

7464 \pgfnodealias % added 2025-04-05
7465 { \@@_env: - last - last \l_@@_suffix_tl }
7466 { \@@_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 .

```

7467 \seq_map_pairwise_function:NNN
7468 \g_@@_multicolumn_cells_seq
7469 \g_@@_multicolumn_sizes_seq
7470 \@@_node_for_multicolumn:nn
7471 }

7472 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7473 {
7474   \cs_set_nopar:Npn \@@_i: { #1 }
7475   \cs_set_nopar:Npn \@@_j: { #2 }
7476 }

```

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”).

```

7477 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7478 {
7479   \@@_extract_coords_values: #1 \q_stop
7480   \@@_pgf_rect_node:nnnnn
7481   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7482   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7483   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7484   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: + #2 - 1 } _ max _ dim } }
7485   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7486   \str_if_empty:NF \l_@@_name_str
7487   {
7488     \pgfnodealias
7489     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7490     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7491   }
7492 }

```

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).

```

7493 \keys_define:nn { nicematrix / Block / FirstPass }
7494 {
7495   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7496           \bool_set_true:N \l_@@_p_block_bool ,
7497   j .value_forbidden:n = true ,
7498   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7499   l .value_forbidden:n = true ,
7500   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7501   r .value_forbidden:n = true ,
7502   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7503   c .value_forbidden:n = true ,

```

```

7504 L.code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7505 L.value_forbidden:n = true ,
7506 R.code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7507 R.value_forbidden:n = true ,
7508 C.code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7509 C.value_forbidden:n = true ,
7510 t.code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7511 t.value_forbidden:n = true ,
7512 T.code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7513 T.value_forbidden:n = true ,
7514 b.code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7515 b.value_forbidden:n = true ,
7516 B.code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7517 B.value_forbidden:n = true ,
7518 m.code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7519 m.value_forbidden:n = true ,
7520 v-center.meta:n = m ,
7521 p.code:n = \bool_set_true:N \l_@@_p_block_bool ,
7522 p.value_forbidden:n = true ,
7523 respect-arraystretch.code:n =
7524   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7525 respect-arraystretch.value_forbidden:n = true ,
7526 color.code:n =
7527   \@@_color:n { #1 }
7528   \tl_set_rescan:Nnn
7529     \l_@@_draw_tl
7530     { \char_set_catcode_other:N ! }
7531     { #1 } ,
7532 color.value_required:n = true ,
7533 }

```

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.

```

7534 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7535 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7536 {

```

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).

```

7537   \tl_if_blank:nTF { #2 }
7538     { \@@_Block_ii:nnnnn 1 1 }
7539     {
7540       \tl_if_in:nnTF { #2 } { - }
7541       {
7542         \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7543         \@@_Block_i_czech:w \@@_Block_i:w
7544         #2 \q_stop
7545       }
7546       {
7547         \@@_error:nn { Bad~argument~for~Block } { #2 }
7548         \@@_Block_ii:nnnnn 1 1
7549       }
7550     }
7551   { #1 } { #3 } { #4 }
7552   \ignorespaces
7553 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7554 \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`.

```

7555 {
7556   \char_set_catcode_active:N -
7557   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7558 }

```

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.

```

7559 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7560 {

```

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).

```

7561   \bool_lazy_or:nnTF
7562     { \tl_if_blank_p:n { #1 } }
7563     { \str_if_eq_p:ee { * } { #1 } }
7564     { \int_set:Nn \l_tmpa_int { 100 } }
7565     { \int_set:Nn \l_tmpa_int { #1 } }
7566   \bool_lazy_or:nnTF
7567     { \tl_if_blank_p:n { #2 } }
7568     { \str_if_eq_p:ee { * } { #2 } }
7569     { \int_set:Nn \l_tmpb_int { 100 } }
7570     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7571   \int_compare:nNnTF \l_tmpb_int = 1
7572   {
7573     \tl_if_empty:NTF \l_@@_hpos_cell_tl
7574       { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7575       { \str_set:Nn \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7576   }
7577   { \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.

```

7578   \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7579   \tl_set:Ne \l_tmpa_tl
7580   {
7581     { \int_use:N \c@iRow }
7582     { \int_use:N \c@jCol }
7583     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7584     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7585   }

```

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).

```

7586   \bool_set_false:N \l_tmpa_bool
7587   \bool_if:NT \l_@@_amp_in_blocks_bool

```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

7588     { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7589     \bool_case:nF
7590     {
7591         \l_tmpa_bool                { \@@_Block_vii:eennn }
7592         \l_@@_p_block_bool          { \@@_Block_vi:eennn }

```

For the blocks mono-column, we will compose right away in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a X column, we should not do that since the width is determined by another way. This should be the same for the p, m and b columns and we should modify that point. However, for the X column, it's imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

7593         \l_@@_X_bool                { \@@_Block_v:eennn }
7594         { \tl_if_empty_p:n { #5 } }   { \@@_Block_v:eennn }
7595         { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7596         { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7597     }
7598     { \@@_Block_v:eennn }
7599     { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7600 }

```

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.

```

7601 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7602 {
7603     \int_gincr:N \g_@@_block_box_int
7604     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
7605     \cs_set_eq:NN \rowcolor \@@_rowcolor_error
7606     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7607     {
7608         \tl_gput_right:Ne \g_@@_pre_code_after_tl
7609         {
7610             \@@_actually_diagbox:nnnnnn
7611             { \int_use:N \c@iRow }
7612             { \int_use:N \c@jCol }
7613             { \int_eval:n { \c@iRow + #1 - 1 } }
7614             { \int_eval:n { \c@jCol + #2 - 1 } }
7615             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7616             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7617         }
7618     }
7619     \box_gclear_new:c
7620     { g_@@_block_box_int _ \int_use:N \g_@@_block_box_int _ box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7621     \hbox_gset:cn
7622     { g_@@_block_box_int _ \int_use:N \g_@@_block_box_int _ box }
7623     {

```

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`).

```

7624      \tl_if_empty:NTF \l_@@_color_tl
7625      { \int_compare:nNnT { #2 } = 1 { \set@color } }
7626      { \@@_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`.

```

7627      \int_compare:nNnT { #1 } = 1
7628      {
7629          \int_if_zero:nTF \c@iRow
7630          {

```

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}$

```

```

7631      \cs_set_eq:NN \Block \@@_NullBlock:
7632      \l_@@_code_for_first_row_tl
7633      }
7634      {
7635          \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7636          {
7637              \cs_set_eq:NN \Block \@@_NullBlock:
7638              \l_@@_code_for_last_row_tl
7639          }
7640      }
7641      \g_@@_row_style_tl
7642      }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7643      \@@_reset_arraystretch:
7644      \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7645      #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.).

```

7646      \@@_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}`.

```

7647 \bool_if:NTF \l_@@_tabular_bool
7648 {
7649   \bool_lazy_all:nTF
7650   {
7651     { \int_compare_p:nNn { #2 } = 1 }

```

Remind that, when the column has not a fixed width, the dimension `\l_@@_col_width_dim` has the conventional value of -1 cm.

```

7652     { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7653     { ! \g_@@_rotate_bool }
7654   }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7655   {
7656     \use:e
7657     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7658       \exp_not:N \begin { minipage }
7659       [ \str_lowercase:f \l_@@_vpos_block_str ]
7660       { \l_@@_col_width_dim }
7661       \str_case:on \l_@@_hpos_block_str
7662       { c \centering r \raggedleft l \raggedright }
7663     }
7664     #5
7665     \end { minipage }
7666   }

```

In the other cases, we use a `{tabular}`.

```

7667   {
7668     \use:e
7669     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7670       \exp_not:N \begin { tabular }
7671       [ \str_lowercase:f \l_@@_vpos_block_str ]
7672       { @ { } \l_@@_hpos_block_str @ { } }
7673     }
7674     #5
7675     \end { tabular }
7676   }
7677 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is `false`). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7678   {
7679     $ % $
7680     \use:e
7681     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7682       \exp_not:N \begin { array }
7683       [ \str_lowercase:f \l_@@_vpos_block_str ]
7684       { @ { } \l_@@_hpos_block_str @ { } }
7685     }
7686     #5
7687     \end { array }
7688     $ % $
7689   }
7690 }

```

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).

```

7691 \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.

```

7692     \int_compare:nNnT { #2 } = 1
7693     {
7694         \dim_gset:Nn \g_@@_blocks_wd_dim
7695         {
7696             \dim_max:nn
7697             \g_@@_blocks_wd_dim
7698             {
7699                 \box_wd:c
7700                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7701             }
7702         }
7703     }

```

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.

```

7704     \int_compare:nNnT { #1 } = 1
7705     {
7706         \bool_lazy_any:nT
7707         {
7708             { \str_if_empty_p:N \l_@@_vpos_block_str }
7709             { \str_if_eq_p:ee t \l_@@_vpos_block_str }
7710             { \str_if_eq_p:ee b \l_@@_vpos_block_str }
7711         }
7712         { \@@_adjust_blocks_ht_dp: }
7713     }
7714     \seq_gput_right:Ne \g_@@_blocks_seq
7715     {
7716         \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.

```

7717     {
7718         \exp_not:n { #3 } ,
7719         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7720         \bool_if:NT \g_@@_rotate_bool
7721         {
7722             \bool_if:NTF \g_@@_rotate_c_bool
7723             { m }
7724             {
7725                 \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7726                 { T }
7727             }
7728         }
7729     }
7730     {
7731         \box_use_drop:c
7732         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7733     }
7734 }
7735 \bool_set_false:N \g_@@_rotate_c_bool
7736 }

7737 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7738 {
7739     \dim_gset:Nn \g_@@_blocks_ht_dim
7740     {

```

```

7741     \dim_max:nn
7742     \g_@@_blocks_ht_dim
7743     {
7744         \box_ht:c
7745         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7746     }
7747 }
7748 \dim_gset:Nn \g_@@_blocks_dp_dim
7749 {
7750     \dim_max:nn
7751     \g_@@_blocks_dp_dim
7752     {
7753         \box_dp:c
7754         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7755     }
7756 }
7757 }

7758 \cs_new:Npn \@@_adjust_hpos_rotate:
7759 {
7760     \bool_if:NT \g_@@_rotate_bool
7761     {
7762         \str_set:Nc \l_@@_hpos_block_str
7763         {
7764             \bool_if:NTF \g_@@_rotate_c_bool
7765             c
7766             {
7767                 \str_case:onF \l_@@_vpos_block_str
7768                 { b l B l t r T r }
7769                 {
7770                     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
7771                     r
7772                     l
7773                 }
7774             }
7775         }
7776     }
7777 }
7778 \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*.

```

7779 \cs_new_protected:Npn \@@_rotate_box_of_block:
7780 {
7781     \box_grotate:cn
7782     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7783     { 90 }
7784     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7785     {
7786         \vbox_gset_top:cn
7787         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7788         {
7789             \skip_vertical:n { 0.8 ex }
7790             \box_use:c
7791             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7792         }
7793     }
7794     \bool_if:NT \g_@@_rotate_c_bool
7795     {
7796         \hbox_gset:cn
7797         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7798         {

```

```

7799         $ % $
7800         \vcenter
7801         {
7802             \box_use:c
7803             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7804         }
7805         $ % $
7806     }
7807 }
7808 }

```

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:nnnnnn`).

#1 is i (the number of rows of the block), **#2** is j (the number of columns of the block), **#3** is the list of *key=values* pairs, **#4** are the tokens to put before the math mode and before the composition of the block and **#5** is the label (=content) of the block.

```

7809 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5
7810 {
7811     \seq_gput_right:Ne \g_@@_blocks_seq
7812     {
7813         \l_tmpa_tl
7814         { \exp_not:n { #3 } }
7815         {
7816             \bool_if:NTF \l_@@_tabular_bool
7817             {
7818                 \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7819         \@@_reset_arraystretch:
7820         \exp_not:n
7821         {
7822             \dim_zero:N \extrarowheight
7823             #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).

```

7824         \tag_if_active:T { \tag_stop:n { table } }
7825         \use:e
7826         {
7827             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7828             { @ { } \l_@@_hpos_block_str @ { } }
7829         }
7830         #5
7831         \end { tabular }
7832     }
7833     \group_end:
7834 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7835     {
7836         \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7837         \@@_reset_arraystretch:
7838         \exp_not:n
7839         {
7840             \dim_zero:N \extrarowheight
7841             #4
7842             $ % $

```

```

7843         \use:e
7844         {
7845             \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7846             { @ { } \l_@@_hpos_block_str @ { } }
7847         }
7848         #5
7849         \end { array }
7850         $ % $
7851     }
7852     \group_end:
7853 }
7854 }
7855 }
7856 }
7857 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7858 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7859 {
7860     \seq_gput_right:Ne \g_@@_blocks_seq
7861     {
7862         \l_tmpa_tl
7863         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7864         { { \exp_not:n { #4 #5 } } }
7865     }
7866 }
7867 \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`.

```

7868 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7869 {
7870     \seq_gput_right:Ne \g_@@_blocks_seq
7871     {
7872         \l_tmpa_tl
7873         { \exp_not:n { #3 } }
7874         { \exp_not:n { #4 #5 } }
7875     }
7876 }
7877 \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).

```

7878 \keys_define:nn { nicematrix / Block / SecondPass }
7879 {
7880     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7881     ampersand-in-blocks .default:n = true ,
7882     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7883     tikz .code:n =
7884         \IfPackageLoadedTF { tikz }
7885         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7886         { \@@_error:n { tikz~key~without~tikz } } ,
7887     tikz .value_required:n = true ,
7888     fill .code:n =
7889         \tl_set_rescan:Nnn
7890         \l_@@_fill_tl
7891         { \char_set_catcode_other:N ! }
7892         { #1 } ,
7893     fill .value_required:n = true ,

```

In fine, the opacity will be applied by `\pgfsetfillopacity`.

```

7894 opacity .tl_set:N = \l_@@_opacity_tl ,
7895 opacity .value_required:n = true ,
7896 draw .code:n =
7897     \tl_set_rescan:Nnn
7898     \l_@@_draw_tl
7899     { \char_set_catcode_other:N ! }
7900     { #1 } ,
7901 draw .default:n = default ,
7902 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7903 rounded-corners .default:n = 4 pt ,
7904 color .code:n =
7905     \@@_color:n { #1 }
7906     \tl_set_rescan:Nnn
7907     \l_@@_draw_tl
7908     { \char_set_catcode_other:N ! }
7909     { #1 } ,
7910 borders .clist_set:N = \l_@@_borders_clist ,
7911 borders .value_required:n = true ,
7912 hvlines .meta:n = { vlines , hlines } ,
7913 vlines .bool_set:N = \l_@@_vlines_block_bool ,
7914 vlines .default:n = true ,
7915 hlines .bool_set:N = \l_@@_hlines_block_bool ,
7916 hlines .default:n = true ,
7917 rules/width .dim_set:N = \arrayrulewidth ,
7918 rules/width .value_required:n = true ,

```

The key `line-width` is now deprecated (replaced by `rules/width`).

```

7919 line-width .dim_set:N = \l_@@_line_width_dim ,
7920 line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7921 j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7922     \bool_set_true:N \l_@@_p_block_bool ,
7923 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7924 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7925 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7926 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7927     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7928 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7929     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7930 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7931     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7932 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7933 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7934 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7935 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7936 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7937 m .value_forbidden:n = true ,
7938 v-center .meta:n = m ,
7939 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7940 p .value_forbidden:n = true ,
7941 name .tl_set:N = \l_@@_block_name_str , % .str_set:N ?
7942 name .value_required:n = true ,
7943 name .initial:n = ,
7944 respect-arraystretch .code:n =
7945     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7946 respect-arraystretch .value_forbidden:n = true ,
7947 transparent .bool_set:N = \l_@@_transparent_bool ,
7948 transparent .default:n = true ,
7949 transparent .initial:n = false ,
7950 unknown .code:n =
7951     \@@_unknown_key:nn
7952     { nicematrix / Block / SecondPass }

```

```

7953     { Unknown-key-for-Block }
7954 }

```

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.

```

7955 \cs_new_protected:Npn \@@_draw_blocks:
7956 {
7957   \bool_if:NTF \c_@@_revtex_bool
7958     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7959     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7960   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7961 }
7962 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7963 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7964   \int_zero:N \l_@@_last_row_int
7965   \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”).

```

7966   \int_compare:nNnTF { #3 } > { 98 }
7967     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7968     { \int_set:Nn \l_@@_last_row_int { #3 } }
7969   \int_compare:nNnTF { #4 } > { 98 }
7970     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7971     { \int_set:Nn \l_@@_last_col_int { #4 } }
7972   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7973   {
7974     \bool_lazy_and:nnTF
7975       \l_@@_preamble_bool
7976       {
7977         \int_compare_p:n
7978           { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7979       }
7980     {
7981       \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7982       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7983       \@@_msg_redirect_name:nn { columns-not-used } { none }
7984     }
7985     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7986   }
7987   {
7988     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7989       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7990       {
7991         \@@_Block_v:nneenn
7992           { #1 }
7993           { #2 }
7994           { \int_use:N \l_@@_last_row_int }
7995           { \int_use:N \l_@@_last_col_int }
7996           { #5 }
7997           { #6 }
7998       }
7999   }
8000 }

```

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.

```
8001 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
8002 {
```

The group is for the keys.

```
8003   \group_begin:
8004   \int_compare:nNtT { #1 } = { #3 }
8005     { \str_set:Nn \l_@@_vpos_block_str { t } }
8006   \keys_set:nn { nicematrix / Block / SecondPass } { #5 }
```

If the content of the block contains `&`, we will have a special treatment (since the cell must be divided in several sub-cells). Remark that `\tl_if_in:nnT` is faster than `\str_if_in:nnT`.

```
8007   \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
8008   \bool_lazy_and:nnT
8009     \l_@@_vlines_block_bool
8010     { ! \l_@@_ampersand_bool }
8011   {
8012     \tl_gput_right:Ne \g_nicematrix_code_after_tl
8013     {
8014       \@@_vlines_block:nnnnn
8015       { \dim_use:N \arrayrulewidth }
8016       { #1 } { #2 } { #3 } { #4 }
8017     }
8018   }
8019   \bool_if:NT \l_@@_hlines_block_bool
8020   {
8021     \tl_gput_right:Ne \g_nicematrix_code_after_tl
8022     {
8023       \@@_hlines_block:nnnnn
8024       { \dim_use:N \arrayrulewidth }
8025       { #1 } { #2 } { #3 } { #4 }
8026     }
8027   }
```

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).

```
8028   \bool_if:NF \l_@@_transparent_bool
8029   {
8030     \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
8031     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
8032   }

8033   \tl_if_empty:NF \l_@@_draw_tl
8034   {
8035     \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
8036     { \@@_error:n { hlines~with~color } }
8037     \tl_gput_right:Nn \g_nicematrix_code_after_tl
8038     {
8039       \@@_stroke_block:nnnnn
```

#5 are the options

```
8040       { #5 } { #1 } { #2 } { #3 } { #4 }
8041     }
8042     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
8043     { { #1 } { #2 } { #3 } { #4 } }
8044   }

8045   \clist_if_empty:NF \l_@@_borders_clist
8046   {
8047     \tl_gput_right:Nn \g_nicematrix_code_after_tl
8048     {
8049       \@@_stroke_borders_block:nnnnn
```

```

8050         { #5 } { #1 } { #2 } { #3 } { #4 }
8051     }
8052 }
8053 \tl_if_empty:NF \l_@@_fill_tl
8054 {
8055     \@@_add_opacity_to_fill:
8056     \tl_gput_right:Ne \g_@@_pre_code_before_tl
8057     {
8058         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
8059         { #1 - #2 }
8060         { #3 - #4 }
8061         { \dim_use:N \l_@@_rounded_corners_dim }
8062     }
8063 }
8064 \seq_if_empty:NF \l_@@_tikz_seq
8065 {
8066     \tl_gput_right:Ne \g_nicematrix_code_before_tl
8067     {
8068         \@@_block_tikz:nnnnn
8069         { \seq_use:Nn \l_@@_tikz_seq { , } }
8070         { #1 } { #2 } { #3 } { #4 }

```

We will have in that last field a list of lists of TikZ keys.

```

8071     }
8072 }
8073 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
8074 {
8075     \tl_gput_right:Ne \g_@@_pre_code_after_tl
8076     {
8077         \@@_actually_diagbox:nnnnn
8078         { #1 } { #2 } { #3 } { #4 }
8079         { \exp_not:n { ##1 } }
8080         { \exp_not:n { ##2 } }
8081     }
8082 }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} &      & one      & \\
                        &      & two      & \\
three                  & four & five     & \\
six                    & seven & eight    & \\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block		one
three	four	two
six	seven	five
		eight

We highlight the node `1-1-block-short`

our block		one
three	four	two
six	seven	five
		eight

The construction of the node corresponding to the merged cells.

```

8083 \pgfpicture
8084 \pgfrememberpicturepositiononpagetrue
8085 \pgf@relevantforpicturesizefalse
8086 \@@_qpoint:n { row - #1 }

```



```

8087 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8088 \@@_qpoint:n { col - #2 }
8089 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8090 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
8091 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8092 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8093 \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.

```

8094 \@@_pgf_rect_node:nnnnn
8095 { \@@_env: - #1 - #2 - block }
8096 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8097 \str_if_empty:NF \l_@@_block_name_str
8098 {
8099   \pgfnodealias
8100   { \@@_env: - \l_@@_block_name_str }
8101   { \@@_env: - #1 - #2 - block }
8102   \str_if_empty:NF \l_@@_name_str
8103   {
8104     \pgfnodealias
8105     { \l_@@_name_str - \l_@@_block_name_str }
8106     { \@@_env: - #1 - #2 - block }
8107   }
8108 }

```

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.

```

8109 \bool_if:NF \l_@@_hpos_of_block_cap_bool
8110 {
8111   \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.

```

8112 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8113 {

```

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.

```

8114 \cs_if_exist:cT
8115 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8116 {
8117   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8118   {
8119     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
8120     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
8121   }
8122 }
8123 }

```

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.

```

8124 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
8125 {
8126   \@@_qpoint:n { col - #2 }
8127   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8128 }
8129 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
8130 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8131 {
8132   \cs_if_exist:cT

```

```

8133     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8134     {
8135       \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8136       {
8137         \pgfpointanchor
8138         { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8139         { east }
8140         \dim_set:Nn \l_@@_tmpd_dim
8141         { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
8142       }
8143     }
8144   }
8145   \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
8146   {
8147     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8148     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8149   }
8150   \@@_pgf_rect_node:nnnnn
8151   { \@@_env: - #1 - #2 - block - short }
8152   \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8153 }

```

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.

```

8154   \bool_if:NT \l_@@_medium_nodes_bool
8155   {
8156     \@@_pgf_rect_node:nnn
8157     { \@@_env: - #1 - #2 - block - medium }
8158     { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
8159     {
8160       \pgfpointanchor
8161       { \@@_env:
8162         - \int_use:N \l_@@_last_row_int
8163         - \int_use:N \l_@@_last_col_int - medium
8164       }
8165       { south-east }
8166     }
8167   }
8168   \endpgfpicture
8169

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand `&`.

```

8170   \bool_if:NTF \l_@@_ampersand_bool
8171   {
8172     \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
8173     \int_zero_new:N \l_@@_split_int
8174     \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. We use locally counters that have another signification in the main environment (maybe we should change that).

```

8175     \int_set:Nn \l_@@_first_row_int { #1 }
8176     \int_set:Nn \l_@@_first_col_int { #2 }
8177     \int_set:Nn \l_@@_last_row_int { #3 }
8178     \int_set:Nn \l_@@_last_col_int { #4 }
8179   \pgfpicture
8180   \pgfrememberpicturepositiononpagetrue
8181   \pgf@relevantforpicturesizefalse
8182   \@@_qpoint:n { row - #1 }
8183   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8184   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8185   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
8186   \@@_qpoint:n { col - #2 }

```

```

8187 \dim_set_eq:NN \l_tmpa_dim \pgf@x
8188 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8189 \dim_set:Nn \l_tmpb_dim
8190 { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
8191 \bool_lazy_or:nnT
8192 \l_@@_vlines_block_bool
8193 { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
8194 {
8195   \int_step_inline:nn { \l_@@_split_int - 1 }
8196   {
8197     \pgfpathmoveto
8198     {
8199       \pgfpoint
8200       { \l_tmpa_dim + ##1 \l_tmpb_dim }
8201       \l_@@_tmpc_dim
8202     }
8203     \pgfpathlineto
8204     {
8205       \pgfpoint
8206       { \l_tmpa_dim + ##1 \l_tmpb_dim }
8207       \l_@@_tmpd_dim
8208     }
8209     \CT@arc@
8210     \pgfsetlinewidth { 1.1 \arrayrulewidth }
8211     \pgfsetrectcap
8212     \pgfusepathqstroke
8213   }
8214 }
8215 \cs_set_eq:NN \cellcolor \@@_subcellcolor
8216 \int_zero_new:N \l_@@_split_i_int
8217 \str_if_eq:eeTF \l_@@_vpos_block_str T
8218 {
8219   \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
8220   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8221 }
8222 {
8223   \str_if_eq:eeTF \l_@@_vpos_block_str B
8224   {
8225     \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
8226     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8227   }
8228   {
8229     \bool_lazy_or:nnTF
8230     { \int_compare_p:nNn { #1 } = { #3 } }
8231     { \str_if_eq_p:ee \l_@@_vpos_block_str t }
8232     {
8233       \@@_qpoint:n { row - #1 - base }
8234       \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8235     }
8236     {
8237       \str_if_eq:eeTF \l_@@_vpos_block_str b
8238       {
8239         \@@_qpoint:n { row - #3 - base }
8240         \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8241       }
8242       {
8243         \@@_qpoint:n { #1 - #2 - block }
8244         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8245       }
8246     }
8247   }
8248 }
8249 \int_step_inline:nn \l_@@_split_int

```

```

8250 {
8251     \group_begin:
The counter \l_@@_split_i_int is only for the command \l_@@_subcellcolor.
8252     \int_set:Nn \l_@@_split_i_int { ##1 }
8253     \dim_set:Nn \col@sep
8254     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
8255     \pgftransformshift
8256     {
8257         \pgfpoint
8258         {
8259             \l_tmpa_dim + ##1 \l_tmpb_dim -
8260             \str_case:on \l_@@_hpos_block_str
8261             {
8262                 l { \l_tmpb_dim + \col@sep }
8263                 c { 0.5 \l_tmpb_dim }
8264                 r { \col@sep }
8265             }
8266         }
8267         { \l_@@_tmpc_dim }
8268     }
8269     \pgfset { inner~sep = \c_zero_dim }
8270     \pgfnode
8271     { rectangle }
8272     {
8273         \str_if_eq:eeTF T \l_@@_vpos_block_str
8274         {
8275             \str_case:on \l_@@_hpos_block_str
8276             {
8277                 l { north-west }
8278                 c { north }
8279                 r { north-east }
8280             }
8281         }
8282         {
8283             \str_if_eq:eeTF B \l_@@_vpos_block_str
8284             {
8285                 \str_case:on \l_@@_hpos_block_str
8286                 {
8287                     l { south-west }
8288                     c { south }
8289                     r { south-east }
8290                 }
8291             }
8292             {
8293                 \bool_lazy_any:nTF
8294                 {
8295                     { \int_compare_p:nNn { #1 } = { #3 } }
8296                     { \str_if_eq_p:ee t \l_@@_vpos_block_str }
8297                     { \str_if_eq_p:ee b \l_@@_vpos_block_str }
8298                 }
8299                 {
8300                     \str_case:on \l_@@_hpos_block_str
8301                     {
8302                         l { base-west }
8303                         c { base }
8304                         r { base-east }
8305                     }
8306                 }
8307             }
8308             {
8309                 \str_case:on \l_@@_hpos_block_str
8310                 {
8311                     l { west }
8312                     c { center }

```

```

8312             r { east }
8313         }
8314     }
8315 }
8316 }
8317 }
8318 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8319 \group_end:
8320 }
8321 \endpgfpicture
8322 }

```

Now the case where there is no ampersand & in the content of the block.

```

8323 {
8324   \bool_if:NTF \l_@@_p_block_bool
8325   {

```

When the final user has used the key p, we have to compute the width.

```

8326   \pgfpicture
8327   \pgfrememberpicturepositiononpagetrue
8328   \pgf@relevantforpicturesizefalse
8329   \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8330   {
8331     \@@_qpoint:n { col - #2 }
8332     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8333     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8334   }
8335   {
8336     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8337     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8338     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8339   }
8340   \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8341 \endpgfpicture
8342 \hbox_set:Nn \l_@@_cell_box
8343 {
8344   \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8345     { \g_tmpb_dim }
8346     \str_case:on \l_@@_hpos_block_str
8347     { c \centering r \raggedleft l \raggedright j { } }
8348     #6
8349     \end { minipage }
8350   }
8351 }
8352 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
8353 \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.

```

8354   \pgfpicture
8355   \pgfrememberpicturepositiononpagetrue
8356   \pgf@relevantforpicturesizefalse
8357   \bool_lazy_any:nTF
8358   {
8359     { \str_if_empty_p:N \l_@@_vpos_block_str }
8360     { \str_if_eq_p:ee c \l_@@_vpos_block_str }
8361     { \str_if_eq_p:ee T \l_@@_vpos_block_str }
8362     { \str_if_eq_p:ee B \l_@@_vpos_block_str }
8363   }
8364   {

```

If we are in the “first column”, we must put the block as if it was with the key r.

```

8365     \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 1.

```

8366     \bool_if:nT \g_@@_last_col_found_bool
8367     {
8368         \int_compare:nNnT { #2 } = \g_@@_col_total_int
8369         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_1_str }
8370     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8371     \tl_set:Ne \l_tmpa_tl
8372     {
8373         \str_case:on \l_@@_vpos_block_str
8374         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8375         { } {
8376             \str_case:on \l_@@_hpos_block_str
8377             {
8378                 c { center }
8379                 l { west }
8380                 r { east }
8381                 j { center }
8382             }
8383         }
8384     c {
8385         \str_case:on \l_@@_hpos_block_str
8386         {
8387             c { center }
8388             l { west }
8389             r { east }
8390             j { center }
8391         }
8392     }
8393 }
8394 T {
8395     \str_case:on \l_@@_hpos_block_str
8396     {
8397         c { north }
8398         l { north-west }
8399         r { north-east }
8400         j { north }
8401     }
8402 }
8403 }
8404 B {
8405     \str_case:on \l_@@_hpos_block_str
8406     {
8407         c { south }
8408         l { south-west }
8409         r { south-east }
8410         j { south }
8411     }
8412 }
8413 }
8414 }
8415 }
8416 \pgftransformshift
8417 {
8418     \pgfpointanchor
8419     {
8420         \@@_env: - #1 - #2 - block
8421         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8422     }
8423     \l_tmpa_tl

```

```

8424     }
8425     \pgfset { inner~sep = \c_zero_dim }
8426     \pgfnode
8427     { rectangle }
8428     \l_tmpa_tl
8429     { \box_use_drop:N \l_@@_cell_box } { } { }
8430 }

```

End of the case when `\l_@@_vpos_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```

8431 {
8432     \pgfextracty \l_tmpa_dim
8433     {
8434         \@@_qpoint:n
8435         {
8436             row - \str_if_eq:eeTF b \l_@@_vpos_block_str { #3 } { #1 }
8437             - base
8438         }
8439     }
8440     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

8441     \pgfpointanchor
8442     {
8443         \@@_env: - #1 - #2 - block
8444         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8445     }
8446     {
8447         \str_case:on \l_@@_hpos_block_str
8448         {
8449             c { center }
8450             l { west }
8451             r { east }
8452             j { center }
8453         }
8454     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8455     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8456     \pgfset { inner~sep = \c_zero_dim }
8457     \pgfnode
8458     { rectangle }
8459     {
8460         \str_case:on \l_@@_hpos_block_str
8461         {
8462             c { base }
8463             l { base~west }
8464             r { base~east }
8465             j { base }
8466         }
8467     }
8468     { \box_use_drop:N \l_@@_cell_box } { } { }
8469 }
8470 \endpgfpicture
8471 }
8472 \group_end:
8473 }
8474 \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).

```

8475 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8476 {

```

```

8477 \tl_if_empty:NF \l_@@_opacity_tl
8478 {
8479   \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8480     {
8481       \tl_set:Ne \l_@@_fill_tl
8482       {
8483         [ opacity = \l_@@_opacity_tl ,
8484           \tl_tail:o \l_@@_fill_tl
8485         ]
8486       }
8487     }
8488     {
8489       \tl_set:Ne \l_@@_fill_tl
8490       { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8491     }
8492   }

```

The first argument of `\@@_stroke_block:nnn` is a list of options for the rectangle that you will stroke. The other arguments are the position of the block: *imin*, *jmin*, *imax* and *jmax*.

```

8493 \cs_new_protected:Npn \@@_stroke_block:nnnnn #1 #2 #3 #4 #5
8494 {
8495   \group_begin:
8496   \tl_clear:N \l_@@_draw_tl
8497   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8498   \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8499   \pgfpicture
8500   \pgfrememberpicturepositiononpagetrue
8501   \pgf@relevantforpicturesizefalse
8502   \tl_if_empty:NF \l_@@_draw_tl
8503   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8504     \tl_if_eq:NnTF \l_@@_draw_tl { default }
8505     \CT@arc@
8506     { \@@_color:o \l_@@_draw_tl }
8507   }

```

The following code can't be put just before the `\pgfusepath { stroke }` (it's too late).

```

8508 \pgfsetcornersarced
8509 { \pgfpoint \l_@@_rounded_corners_dim \l_@@_rounded_corners_dim }
8510 \int_compare:nNnF { #2 } > \c@iRow
8511 {
8512   \int_compare:nNnF { #3 } > \c@jCol
8513   {
8514     \@@_qpoint:n { row - #2 }
8515     \dim_set_eq:NN \l_tmpb_dim \pgf@y
8516     \@@_qpoint:n { col - #3 }
8517     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8518     \@@_qpoint:n
8519     { row - \int_eval:n { 1 + \int_min:nn \c@iRow { #4 } } }
8520     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8521     \@@_qpoint:n
8522     { col - \int_eval:n { 1 + \int_min:nn \c@jCol { #5 } } }
8523     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8524     \pgfpathrectanglecorners
8525     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8526     { \pgfpoint \pgf@x \l_tmpa_dim }
8527     \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
8528     \pgfusepathqstroke
8529     { \pgfusepath { stroke } }
8530   }
8531 }
8532 \endpgfpicture

```



```

8533 \group_end:
8534 }

```

Here is the set of keys for the command `\@@_stroke_block:nnnnn`.

```

8535 \keys_define:nn { nicematrix / BlockStroke }
8536 {
8537   color .tl_set:N = \l_@@_draw_tl ,
8538   draw .code:n =
8539     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8540   draw .default:n = default ,
8541   line-width .dim_set:N = \l_@@_line_width_dim ,
8542   rules/width .dim_set:N = \l_@@_line_width_dim ,
8543   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8544   rounded-corners .default:n = 4 pt
8545 }

```

The command `\@@_vlines_block:nnnnn` is used only once: in `\@@_Block_v:nnnnnn` which actually draw the block.

The first argument of `\@@_vlines_block:nnn` is the width of the rules that we will draw. The other arguments are the position of the block: *imin*, *jmin*, *imax* and *jmax*.

```

8546 \cs_new_protected:Npn \@@_vlines_block:nnnnn #1 #2 #3 #4 #5
8547 {
8548   \group_begin:

```

We are actually during the execution of the `\g_nicematrix_code_after_tl`. In that context `\arrayrulewidth` is the width of standard lines (we are drawing standard rules because, up to now, there is no way to draw the internal lines of a Block with a style of TikZ).

```

8549   \dim_set:Nn \arrayrulewidth { #1 }

```

Below, you use `\@@_vline:n` to draw the rules and that command will use `\g_@@_pos_of_blocks_seq` in order to *not* draw the rules within the blocks. That's why we filter the list of blocks in order to discard the blocks which encompass the current block.

```

8550   \seq_set_filter:Nn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
8551   {
8552     \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #2 }
8553     ||
8554     \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #3 }
8555     ||
8556     \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #4 }
8557     ||
8558     \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #5 }
8559   }
8560   \int_step_inline:nnn { #3 } { #5 + 1 }
8561   {
8562     \@@_vline:n
8563     {
8564       position = ##1 ,
8565       start = #2 ,
8566       end = #4 ,
8567     }
8568   }
8569   \group_end:
8570 }

```

The command `\@@_hlines_block:nnnnn` is used only once: in `\@@_Block_v:nnnnnn` which actually draw the block.

```

8571 \cs_new_protected:Npn \@@_hlines_block:nnnnn #1 #2 #3 #4 #5
8572 {

```

We are actually during the execution of the `\g_nicematrix_code_after_tl`. In that context `\arrayrulewidth` is the width of standard lines (we are drawing standard rules because, up to now, there is no way to draw the internal lines of a Block with a style of TikZ).

```
8573 \group_begin:
8574 \dim_set:Nn \arrayrulewidth { #1 }
```

Below, you use `\@@_hline:n` to draw the rules and that command will use `\g_@@_pos_of_blocks_seq` in order to *not* draw the rules within the blocks. That's why we filter the list of blocks in order to discard the blocks which encompass the current block.

```
8575 \seq_set_filter:NnN \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
8576 {
8577   \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #2 }
8578   ||
8579   \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #3 }
8580   ||
8581   \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #4 }
8582   ||
8583   \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #5 }
8584 }
8585 \int_step_inline:nnn { #2 } { #4 + 1 }
8586 {
8587   \@@_hline:n
8588   {
8589     position = ##1 ,
8590     start = #3 ,
8591     end = #5
8592   }
8593 }
8594 \group_end:
8595 }
```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke.

```
8596 \cs_new_protected:Npn \@@_stroke_borders_block:nnnnn #1 #2 #3 #4 #5
8597 {
8598   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8599   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8600   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8601   { \@@_error:n { borders~forbidden } }
8602   {
8603     \tl_clear_new:N \l_@@_borders_tikz_tl
8604     \keys_set:no { nicematrix / OnlyForTikzInBorders } \l_@@_borders_clist
8605     \tl_set:Nn \l_@@_tmpc_tl { #2 }
8606     \tl_set:Nn \l_@@_tmpd_tl { #3 }
8607     \tl_set:Nc \l_@@_tmpa_tl { \int_eval:n { #4 + 1 } }
8608     \tl_set:Nc \l_@@_tmpb_tl { \int_eval:n { #5 + 1 } }
8609     \@@_stroke_borders_block_i:
8610   }
8611 }
8612 \hook_gput_code:nnn { begindocument } { . }
8613 {
8614   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8615   {
8616     \c_@@_pgfortikzpicture_tl
8617     \@@_stroke_borders_block_ii:
8618     \c_@@_endpgfortikzpicture_tl
8619   }
8620 }
8621 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8622 {
8623   \pgfrememberpicturepositiononpagetrue
```

```

8624 \pgf@relevantforpicturesizefalse
8625 \CT@arc@
8626 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8627 \clist_if_in:NnT \l_@@_borders_clist { right }
8628 { \@@_stroke_vertical:n \l_tmpb_tl }
8629 \clist_if_in:NnT \l_@@_borders_clist { left }
8630 { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8631 \clist_if_in:NnT \l_@@_borders_clist { bottom }
8632 { \@@_stroke_horizontal:n \l_tmpa_tl }
8633 \clist_if_in:NnT \l_@@_borders_clist { top }
8634 { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8635 }
8636 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8637 {
8638   tikz .code:n =
8639     \cs_if_exist:NTF \tikzpicture
8640     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8641     { \@@_error:n { tikz~in~borders~without~tikz } } ,
8642   tikz .value_required:n = true ,
8643   top .code:n = ,
8644   bottom .code:n = ,
8645   left .code:n = ,
8646   right .code:n = ,
8647   unknown .code:n = \@@_error:n { bad~border }
8648 }

```

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).

```

8649 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8650 {
8651   \@@_qpoint:n \l_@@_tmpc_tl
8652   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8653   \@@_qpoint:n \l_tmpa_tl
8654   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8655   \@@_qpoint:n { #1 }
8656   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8657   {
8658     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8659     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8660     \pgfusepath{stroke}
8661   }
8662   {
8663     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8664     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8665   }
8666 }

```

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).

```

8667 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8668 {
8669   \@@_qpoint:n \l_@@_tmpd_tl
8670   \clist_if_in:NnTF \l_@@_borders_clist { left }
8671   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8672   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8673   \@@_qpoint:n \l_tmpb_tl
8674   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8675   \@@_qpoint:n { #1 }
8676   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8677   {
8678     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8679     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8680     \pgfusepath{stroke}

```

```

8681     }
8682     {
8683         \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8684         ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8685     }
8686 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

8687 \keys_define:nn { nicematrix / BlockBorders }
8688 {
8689     borders .clist_set:N = \l_@@_borders_clist ,
8690     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8691     rounded-corners .default:n = 4 pt ,
8692     line-width .dim_set:N = \l_@@_line_width_dim ,
8693     rules/width .dim_set:N = \l_@@_line_width_dim
8694 }

```

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.

```

8695 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8696 {
8697     \begin { tikzpicture }
8698     \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because **#5** is a list of lists.

```

8699     \clist_map_inline:nn { #1 }
8700     {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8701         \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8702         \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8703         (
8704             [
8705                 xshift = \dim_use:N \l_@@_offset_dim ,
8706                 yshift = - \dim_use:N \l_@@_offset_dim
8707             ]
8708             #2 -| #3
8709         )
8710         rectangle
8711         (
8712             [
8713                 xshift = - \dim_use:N \l_@@_offset_dim ,
8714                 yshift = \dim_use:N \l_@@_offset_dim
8715             ]
8716             \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8717         ) ;
8718     }
8719     \end { tikzpicture }
8720 }
8721 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

```

```

8722 \keys_define:nn { nicematrix / SpecialOffset }
8723 { 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.

```
8724 \cs_new_protected:Npn \@@_NullBlock:
8725 { \@@_collect_options:n { \@@_NullBlock_i: } }
8726 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8727 { }
```

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.

```
8728 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
8729 {
8730   \tl_gput_right:Nx \g_@@_pre_code_before_tl
8731   { }
```

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`).

```
8732   \@@_subcellcolor:nnnnnnn
8733   {
8734     \tl_if_blank:nTF { #1 }
8735     { { \exp_not:n { #2 } } }
8736     { [ #1 ] { \exp_not:n { #2 } } }
8737   }
8738   { \int_use:N \l_@@_first_row_int } % first row of the block
8739   { \int_use:N \l_@@_first_col_int } % first column of the block
8740   { \int_use:N \l_@@_last_row_int } % last row of the block
8741   { \int_use:N \l_@@_last_col_int } % last column of the block
8742   { \int_use:N \l_@@_split_int }
8743   { \int_use:N \l_@@_split_i_int }
8744 }
8745 \ignorespaces
8746 }

8747 \cs_new_protected:Npn \@@_subcellcolor:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8748 {
8749   \@@_color_opacity: #1
8750   \pgfpicture
8751   \pgf@relevantforpicturesizefalse
8752   \@@_qpoint:n { col - #3 }
8753   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8754   \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
8755   \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
8756   \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
8757   \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
8758   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8759   \@@_qpoint:n { row - #2 }
8760   \pgfpathrectanglecorners
8761   { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } \l_@@_tmpc_dim }
8762   { \pgfpoint \l_tmpb_dim \pgf@y }
8763   \pgfusepathqfill
8764   \endpgfpicture
8765 }
```

27 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```
8766 \keys_define:nn { nicematrix / Auto }
8767 {
```

```

8768 columns-type .tl_set:N = \l_@@_columns_type_tl ,
8769 columns-type .value_required:n = true ,
8770 l .meta:n = { columns-type = l } ,
8771 r .meta:n = { columns-type = r } ,
8772 c .meta:n = { columns-type = c } ,
8773 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8774 delimiters / color .value_required:n = true ,
8775 delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8776 delimiters / max-width .default:n = true ,
8777 delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8778 delimiters .value_required:n = true ,
8779 rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8780 rounded-corners .default:n = 4 pt
8781 }

8782 \NewDocumentCommand \AutoNiceMatrixWithDelims
8783 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8784 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8785 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8786 {

```

The group is for the protection of the keys.

```

8787 \group_begin:
8788 \keys_set:known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8789 \use:e
8790 {
8791 \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8792 { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8793 [ \exp_not:o \l_tmpa_tl ]
8794 }
8795 \int_if_zero:nT \l_@@_first_row_int
8796 {
8797 \int_if_zero:nT \l_@@_first_col_int { & }
8798 \prg_replicate:nn { #4 - 1 } { & }
8799 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8800 }
8801 \prg_replicate:nn { #3 }
8802 {
8803 \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).

```

8804 \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8805 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8806 }
8807 \int_compare:nNnT \l_@@_last_row_int > { -2 }
8808 {
8809 \int_if_zero:nT \l_@@_first_col_int { & }
8810 \prg_replicate:nn { #4 - 1 } { & }
8811 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8812 }
8813 \end { NiceArrayWithDelims }
8814 \group_end:
8815 }

8816 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
8817 {
8818 \cs_set_protected:cpn { #1 AutoNiceMatrix }
8819 {
8820 \bool_gset_true:N \g_@@_delims_bool
8821 \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8822 \AutoNiceMatrixWithDelims { #2 } { #3 }
8823 }
8824 }

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8825 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8826 {
8827   \group_begin:
8828   \bool_gset_false:N \g_@@_delims_bool
8829   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8830   \group_end:
8831 }

```

28 The redefinition of the command `\dotfill`

```

8832 \cs_set_eq:NN \@@_old_dotfill: \dotfill
8833 \cs_new_protected:Npn \@@_dotfill:
8834 {

```

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}`).

```

8835   \@@_old_dotfill:
8836   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8837 }

```

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`.

```

8838 \cs_new_protected:Npn \@@_dotfill_i:
8839 {
8840   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim
8841     { \@@_old_dotfill: }
8842 }

```

29 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.

```

8843 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8844 {
8845   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8846   {
8847     \@@_actually_diagbox:nnnnnn
8848     { \int_use:N \c@iRow }
8849     { \int_use:N \c@jCol }
8850     { \int_use:N \c@iRow }
8851     { \int_use:N \c@jCol }

```

The expansion done on `\g_@@_row_style_tl` will result in the fact that you take into account the current number of row and number of column.

```

8852     { \g_@@_row_style_tl \exp_not:n { #1 } }
8853     { \g_@@_row_style_tl \exp_not:n { #2 } }
8854   }

```

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`.

```

8855   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8856   {
8857     { \int_use:N \c@iRow }
8858     { \int_use:N \c@jCol }
8859     { \int_use:N \c@iRow }
8860     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8861     { }
8862   }
8863 }

```

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.

```

8864 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8865 {
8866   \pgfpicture
8867   \pgf@relevantforpicturesizefalse
8868   \pgfrememberpicturepositiononpagetrue
8869   \@@_qpoint:n { row - #1 }
8870   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8871   \@@_qpoint:n { col - #2 }
8872   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8873   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8874   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8875   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8876   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8877   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8878   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8879   {

```

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.

```

8880   \CT@arc@
8881   \pgfsetroundcap
8882   \pgfusepathqstroke
8883 }
8884 \pgfset { inner~sep = 1 pt }
8885 \pgfscope
8886 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8887 \pgfnode { rectangle } { south~west }
8888 {
8889   \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8890   \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8891   \end { minipage }
8892 }
8893 { }
8894 { }
8895 \endpgfscope
8896 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8897 \pgfnode { rectangle } { north~east }
8898 {
8899   \begin { minipage } { 20 cm }
8900   \raggedleft
8901   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8902   \end { minipage }
8903 }
8904 { }
8905 { }
8906 \endpgfpicture
8907 }

```


30 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. 90.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
8908 \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 `\`.

```
8909 \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`.

```
8910 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8911 {
8912   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8913   \@@_CodeAfter_iv:n
8914 }
```

We catch the argument of the command `\end` (in `#1`).

```
8915 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8916 {
```

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.

```
8917   \str_if_eq:eeTF \@currenvir { #1 }
8918   { \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`.

```
8919   {
8920     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8921     \@@_CodeAfter_ii:n
8922   }
8923 }
```

31 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.

```
8924 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8925 {
8926   \pgfpicture
8927   \pgfrememberpicturepositiononpagetrue
8928   \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.

```

8929 \@@_qpoint:n { row - 1 }
8930 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8931 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8932 \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).

```

8933 \bool_if:nTF { #3 }
8934 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8935 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8936 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8937 {
8938   \cs_if_exist:cT
8939   { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8940   {
8941     \pgfpointanchor
8942     { \@@_env: - ##1 - #2 }
8943     { \bool_if:nTF { #3 } { west } { east } }
8944     \dim_set:Nn \l_tmpa_dim
8945     {
8946       \bool_if:nTF { #3 }
8947       \dim_min:nn
8948       \dim_max:nn
8949       \l_tmpa_dim
8950       \pgf@x
8951     }
8952   }
8953 }

```

Now we can put the delimiter with a node of PGF.

```

8954 \pgfset { inner~sep = \c_zero_dim }
8955 \dim_zero:N \nulldelimiterspace
8956 \pgftransformshift
8957 {
8958   \pgfpoint
8959   \l_tmpa_dim
8960   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8961 }
8962 \pgfnode
8963 { rectangle }
8964 { \bool_if:nTF { #3 } { east } { west } }
8965 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8966 \nullfont
8967 $ % $
8968 \@@_color:o \l_@@_delimiters_color_tl
8969 \bool_if:nTF { #3 } { \left #1 } { \left . }
8970 \vcenter
8971 {
8972   \nullfont
8973   \hrule \@height
8974   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8975   \@depth \c_zero_dim
8976   \@width \c_zero_dim
8977 }
8978 \bool_if:nTF { #3 } { \right . } { \right #1 }
8979 $ % $
8980 }
8981 { }
8982 { }
8983 \endpgfpicture

```

```
8984 }
```

32 The command \SubMatrix

```
8985 \keys_define:nn { nicematrix / sub-matrix }
8986 {
8987   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8988   extra-height .value_required:n = true ,
8989   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8990   left-xshift .value_required:n = true ,
8991   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8992   right-xshift .value_required:n = true ,
8993   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8994   xshift .value_required:n = true ,
8995   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8996   delimiters / color .value_required:n = true ,
8997   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8998   slim .default:n = true ,
8999   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9000   hlines .default:n = all ,
9001   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9002   vlines .default:n = all ,
9003   hvlines .meta:n = { hlines, vlines } ,
9004   hvlines .value_forbidden:n = true
9005 }
9006 \keys_define:nn { nicematrix }
9007 {
9008   SubMatrix .inherit:n = nicematrix / sub-matrix ,
9009   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9010   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9011   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9012 }
```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```
9013 \keys_define:nn { nicematrix / SubMatrix }
9014 {
9015   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9016   delimiters / color .value_required:n = true ,
9017   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9018   hlines .default:n = all ,
9019   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9020   vlines .default:n = all ,
9021   hvlines .meta:n = { hlines, vlines } ,
9022   hvlines .value_forbidden:n = true ,
9023   name .code:n =
9024     \tl_if_empty:nTF { #1 }
9025     { \@@_error:n { Invalid-name } }
9026     {
9027       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
9028       {
9029         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
9030         { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
9031         {
9032           \str_set:Nn \l_@@_submatrix_name_str { #1 }
9033           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
9034         }
9035       }
9036     { \@@_error:n { Invalid-name } }
9037 }
```

```

9038     name .value_required:n = true ,
9039     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
9040     rules .value_required:n = true ,
9041     code .tl_set:N = \l_@@_code_tl ,
9042     code .value_required:n = true ,
9043     unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
9044 }

9045 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
9046 {
9047     \tl_gput_right:Ne \g_@@_pre_code_after_tl
9048     {
9049         \SubMatrix { #1 } { #2 } { #3 } { #4 }
9050         [
9051             delimiters / color = \l_@@_delimiters_color_tl ,
9052             hlines = \l_@@_submatrix_hlines_clist ,
9053             vlines = \l_@@_submatrix_vlines_clist ,
9054             extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
9055             left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
9056             right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
9057             slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
9058             #5
9059         ]
9060     }
9061     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
9062     \ignorespaces
9063 }

9064 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
9065 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9066 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

9067 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
9068 {
9069     \seq_gput_right:Ne \g_@@_submatrix_seq
9070     {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

9071     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
9072     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
9073     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
9074     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
9075 }
9076 }

```

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).

```

9077 \NewDocumentCommand \@@_compute_i_j:nn
9078 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9079 { \@@_compute_i_j:nnnn #1 #2 }

9080 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
9081 {
9082     \def \l_@@_first_i_tl { #1 }
9083     \def \l_@@_first_j_tl { #2 }
9084     \def \l_@@_last_i_tl { #3 }
9085     \def \l_@@_last_j_tl { #4 }
9086     \tl_if_eq:NnT \l_@@_first_i_tl { last }
9087     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
9088     \tl_if_eq:NnT \l_@@_first_j_tl { last }
9089     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
9090     \tl_if_eq:NnT \l_@@_last_i_tl { last }
9091     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
9092     \tl_if_eq:NnT \l_@@_last_j_tl { last }

```

```

9093 { \tl_set:NV \l_@@_last_j_tl \c@jCol }
9094 }

```

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`.

```

9095 \hook_gput_code:nnn { begindocument } { . }
9096 {
9097   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m O { } E { _ ^ } { { } { } } }
9098   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
9099     { \@@_sub_matrix:nnnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
9100 }
9101 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
9102 {
9103   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

9104 \@@_compute_i_j:nn { #2 } { #3 }
9105 \int_compare:nNt \l_@@_first_i_tl = \l_@@_last_i_tl
9106 { \def \arraystretch { 1 } }
9107 \bool_lazy_or:nnTF
9108 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9109 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9110 { \@@_error:nn { Construct~too~large } { \SubMatrix } }
9111 {
9112   \str_clear_new:N \l_@@_submatrix_name_str
9113   \keys_set:nn { nicematrix / SubMatrix } { #5 }
9114   \pgfpicture
9115   \pgfrememberpicturepositiononpagetrue
9116   \pgf@relevantforpicturesizefalse
9117   \pgfset { inner~sep = \c_zero_dim }
9118   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9119   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curryfication.

```

9120 \bool_if:NTF \l_@@_submatrix_slim_bool
9121 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
9122 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
9123 {
9124   \cs_if_exist:cT
9125   { \pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9126   {
9127     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9128     \dim_compare:nNt \pgf@x < \l_@@_x_initial_dim
9129     { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9130   }
9131   \cs_if_exist:cT
9132   { \pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9133   {

```

```

9134         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9135         \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9136         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9137     }
9138 }
9139 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
9140 { \@@_error:nn { Impossible-delimiter } { left } }
9141 {
9142     \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
9143     { \@@_error:nn { Impossible-delimiter } { right } }
9144     { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
9145 }
9146 \endpgfpicture
9147 }
9148 \group_end:
9149 \ignorespaces
9150 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

9151 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
9152 {
9153     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
9154     \dim_set:Nn \l_@@_y_initial_dim
9155     {
9156         \fp_to_dim:n
9157         {
9158             \pgf@y
9159             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
9160         }
9161     }
9162     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
9163     \dim_set:Nn \l_@@_y_final_dim
9164     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
9165     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
9166     {
9167         \cs_if_exist:cT
9168         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
9169         {
9170             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
9171             \dim_set:Nn \l_@@_y_initial_dim
9172             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
9173         }
9174         \cs_if_exist:cT
9175         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
9176         {
9177             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
9178             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
9179             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
9180         }
9181     }
9182     \dim_set:Nn \l_tmpa_dim
9183     {
9184         \l_@@_y_initial_dim - \l_@@_y_final_dim +
9185         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
9186     }
9187     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the \SubMatrix.

```

9188 \group_begin:
9189 \pgfsetlinewidth { 1.1 \arrayrulewidth }
9190 \@@_set_CTarc:o \l_@@_rules_color_tl
9191 \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`.

```

9192 \seq_map_inline:Nn \g_@@_cols_vlism_seq
9193 {
9194   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
9195   {
9196     \int_compare:nNnT
9197       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
9198     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

9199       \@@_qpoint:n { col - ##1 }
9200       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9201       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9202       \pgfusepathqstroke
9203     }
9204   }
9205 }

```

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.

```

9206 \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
9207 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
9208 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
9209 {
9210   \bool_lazy_and:nnTF
9211     { \int_compare_p:nNn { ##1 } > \c_zero_int }
9212     {
9213       \int_compare_p:nNn
9214         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
9215       {
9216         \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
9217         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9218         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9219         \pgfusepathqstroke
9220       }
9221       { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
9222     }

```

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.

```

9223 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
9224 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
9225 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
9226 {
9227   \bool_lazy_and:nnTF
9228     { \int_compare_p:nNn { ##1 } > \c_zero_int }
9229     {
9230       \int_compare_p:nNn
9231         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
9232       {
9233         \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

9234 \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

9235 \dim_set:Nn \l_tmpa_dim
9236 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9237 \str_case:nn { #1 }
9238 {
9239   ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9240   [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }

```

```

9241         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9242     }
9243     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
We compute in \l_tmpb_dim the  $x$ -value of the right end of the rule.
9244     \dim_set:Nn \l_tmpb_dim
9245     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9246     \str_case:nn { #2 }
9247     {
9248         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9249         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
9250         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9251     }
9252     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9253     \pgfusepathqstroke
9254     \group_end:
9255 }
9256 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { #1 } }
9257 }

```

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).

```

9258     \str_if_empty:NF \l_@@_submatrix_name_str
9259     {
9260         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
9261         \l_@@_x_initial_dim \l_@@_y_initial_dim
9262         \l_@@_x_final_dim \l_@@_y_final_dim
9263     }
9264     \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`.

```

9265     \begin { pgfscope }
9266     \pgftransformshift
9267     {
9268         \pgfpoint
9269         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9270         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9271     }
9272     \str_if_empty:NTF \l_@@_submatrix_name_str
9273     { \@@_node_left:nn #1 { } }
9274     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9275     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9276     \pgftransformshift
9277     {
9278         \pgfpoint
9279         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9280         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9281     }
9282     \str_if_empty:NTF \l_@@_submatrix_name_str
9283     { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9284     {
9285         \@@_node_right:nnnn #2
9286         { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9287     }

```

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`.


```

9288 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9289 \flag_clear_new:N \l_@@_code_flag
9290 \l_@@_code_tl
9291 }

```

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`.

```

9292 \cs_set_eq:NN \@@_old_pgfpointanchor: \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of TikZ nodes which are computed in an expandable way.

The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by currying.

```

9293 \cs_new:Npn \@@_pgfpointanchor:n #1
9294 { \exp_args:Ne \@@_old_pgfpointanchor: { \@@_pgfpointanchor_i:n { #1 } } }

```

First, we must detect whether the argument is of the form `\tikz@pp@name{...}` (the command `\tikz@pp@name` is a command of TikZ that adds the prefix and the suffix of the name. If the name refers to a TikZ node which does not exist, there isn't the wrapper `\tikz@pp@name`).

```

9295 \cs_new:Npn \@@_pgfpointanchor_i:n #1
9296 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }
9297 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9298 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9299 \str_if_empty:nTF { #1 }

```

First, when the name of the name begins with `\tikz@pp@name`.

```

9300 { \@@_pgfpointanchor_iv:w #2 }

```

And now, when there is no `\tikz@pp@name`.

```

9301 { \@@_pgfpointanchor_ii:n { #1 } }
9302 }

```

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`).

```

9303 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
9304 { \@@_pgfpointanchor_ii:n { #1 } }

```

With the command `\@@_pgfpointanchor_ii:n`, we deal with the actual name of the node (without the `\tikz@pp@name`). First, we have to detect whether it is of the form i of the form $i-j$ (with an hyphen).

Remark: It would be possible to test the presence of the hyphen in an expandable way to using `\etl_if_in:nnTF` of the package `etl` but, as of now, we do not load `etl`.

```

9305 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

```

```

9306 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
9307 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9308 \str_if_empty:nTF { #2 }

```

First the case where the argument does *not* contain an hyphen.

```

9309 { \@@_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`).

```
9310 { \@@_pgfpointanchor_iii:w { #1 } #2 }
9311 }
```

The following function is for the case when the name contains an hyphen.

```
9312 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9313 {
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
9314 \@@_env:
9315 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9316 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9317 }
```

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.

```
9318 \tl_const:Nn \c_@@_integers_alist_tl
9319 {
9320 { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9321 { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9322 { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9323 { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9324 }
```

```
9325 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9326 {
```

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`.

```
9327 \str_case:nVTF { #1 } \c_@@_integers_alist_tl
9328 {
9329 \flag_raise:N \l_@@_code_flag
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
9330 \@@_env: -
9331 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9332 { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9333 { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9334 }
9335 {
9336 \str_if_eq:eeTF { #1 } { last }
9337 {
9338 \flag_raise:N \l_@@_code_flag
9339 \@@_env: -
9340 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9341 { \int_eval:n { \l_@@_last_i_tl + 1 } }
9342 { \int_eval:n { \l_@@_last_j_tl + 1 } }
9343 }
9344 { #1 }
9345 }
9346 }
```

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`).

```

9347 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9348 {
9349   \pgfnode
9350   { rectangle }
9351   { east }
9352   {
9353     \nullfont
9354     $ % $
9355     \@@_color:o \l_@@_delimiters_color_tl
9356     \left #1
9357     \vcenter
9358     {
9359       \nullfont
9360       \hrule \@height \l_tmpa_dim
9361               \@depth \c_zero_dim
9362               \@width \c_zero_dim
9363     }
9364     \right .
9365     $ % $
9366   }
9367   { #2 }
9368   { }
9369 }

```

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.

```

9370 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9371 {
9372   \pgfnode
9373   { rectangle }
9374   { west }
9375   {
9376     \nullfont
9377     $ % $
9378     \colorlet { current-color } { . }
9379     \@@_color:o \l_@@_delimiters_color_tl
9380     \left .
9381     \vcenter
9382     {
9383       \nullfont
9384       \hrule \@height \l_tmpa_dim
9385               \@depth \c_zero_dim
9386               \@width \c_zero_dim
9387     }
9388     \right #1
9389     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9390     ^ { \color { current-color } \smash { #4 } }
9391     $ % $
9392   }
9393   { #2 }
9394   { }
9395 }

```

33 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9396 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9397 {
9398   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9399   \ignorespaces
9400 }

9401 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9402 {
9403   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9404   \ignorespaces
9405 }

9406 \keys_define:nn { nicematrix / Brace }
9407 {
9408   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9409   left-shorten .default:n = true ,
9410   left-shorten .value_forbidden:n = true ,
9411   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9412   right-shorten .default:n = true ,
9413   right-shorten .value_forbidden:n = true ,
9414   shorten .meta:n = { left-shorten , right-shorten } ,
9415   shorten .value_forbidden:n = true ,
9416   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9417   yshift .value_required:n = true ,
9418   yshift .initial:n = \c_zero_dim ,
9419   color .tl_set:N = \l_tmpa_tl ,
9420   color .value_required:n = true ,
9421   unknown .code:n =
9422     \@@_unknown_key:nn
9423     { nicematrix / Brace }
9424     { Unknown-key~for~Brace }
9425 }

```

`#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`.

```

9426 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9427 {
9428   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9429 \@@_compute_i_j:nn { #1 } { #2 }
9430 \bool_lazy_or:nnTF
9431 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9432 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9433 {
9434   \str_if_eq:eeTF { #5 } { under }
9435   { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9436   { \@@_error:nn { Construct-too-large } { \OverBrace } }
9437 }
9438 {
9439   \tl_clear:N \l_tmpa_tl
9440   \keys_set:nn { nicematrix / Brace } { #4 }
9441   \tl_if_empty:NF \l_tmpa_tl { \color \l_tmpa_tl }
9442   \pgfpicture
9443   \pgfrememberpicturepositiononpagetrue
9444   \pgf@relevantforpicturesizefalse
9445   \bool_if:NT \l_@@_brace_left_shorten_bool
9446   {

```

```

9447 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9448 \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9449 {
9450   \cs_if_exist:cT
9451   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9452   {
9453     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9454
9455     \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9456     { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9457   }
9458 }
9459 }
9460 \bool_lazy_or:nnT
9461 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9462 { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
9463 {
9464   \@@_qpoint:n { col - \l_@@_first_j_tl }
9465   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9466 }
9467 \bool_if:NT \l_@@_brace_right_shorten_bool
9468 {
9469   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9470   \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9471   {
9472     \cs_if_exist:cT
9473     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9474     {
9475       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9476       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9477       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9478     }
9479   }
9480 }
9481 \bool_lazy_or:nnT
9482 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9483 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
9484 {
9485   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9486   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9487 }
9488 \pgfset { inner~sep = \c_zero_dim }
9489 \str_if_eq:eeTF { #5 } { under }
9490 { \@@_underbrace_i:n { #3 } }
9491 { \@@_overbrace_i:n { #3 } }
9492 \endpgfpicture
9493 }
9494 \group_end:
9495 }

```

The argument is the text to put above the brace.

```

9496 \cs_new_protected:Npn \@@_overbrace_i:n #1
9497 {
9498   \@@_qpoint:n { row - \l_@@_first_i_tl }
9499   \pgftransformshift
9500   {
9501     \pgfpoint
9502     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9503     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9504   }
9505   \pgfnode
9506   { rectangle }
9507   { south }
9508   {

```

```

9509 \vtop
9510 {
9511   \group_begin:
9512   \everycr { }
9513   \halign
9514   {
9515     \hfil ## \hfil \crcr
9516     \bool_if:NTF \l_@@_tabular_bool
9517     { \begin { tabular } { c } #1 \end { tabular } }
9518     { $ \begin { array } { c } #1 \end { array } $ }
9519     \cr
9520     $ % $
9521     \overbrace
9522     {
9523       \hbox_to_wd:nn
9524       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9525       { }
9526     }
9527     $ % $
9528     \cr
9529   }
9530   \group_end:
9531 }
9532 }
9533 { }
9534 { }
9535 }

```

The argument is the text to put under the brace.

```

9536 \cs_new_protected:Npn \@@_underbrace_i:n #1
9537 {
9538   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9539   \pgftransformshift
9540   {
9541     \pgfpoint
9542     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9543     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9544   }
9545   \pgfnode
9546   { rectangle }
9547   { north }
9548   {
9549     \group_begin:
9550     \everycr { }
9551     \vbox
9552     {
9553       \halign
9554       {
9555         \hfil ## \hfil \crcr
9556         $ % $
9557         \underbrace
9558         {
9559           \hbox_to_wd:nn
9560           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9561           { }
9562         }
9563         $ % $
9564         \cr
9565         \bool_if:NTF \l_@@_tabular_bool
9566         { \begin { tabular } { c } #1 \end { tabular } }
9567         { $ \begin { array } { c } #1 \end { array } $ }
9568         \cr
9569       }
9570     }
9571   }

```

```

9571     \group_end:
9572   }
9573   { }
9574   { }
9575 }

```

34 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`).

```

9576 \AddToHook { package / tikz / after }
9577 {
9578   \tikzset
9579   {
9580     nicematrix / brace / .style =
9581     {
9582       decoration = { brace , raise = -0.15 em } ,
9583       decorate ,
9584     } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9585     nicematrix / mirrored-brace / .style =
9586     {
9587       nicematrix / brace ,
9588       decoration = mirror ,
9589     }
9590   }
9591 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9592 \keys_define:nn { nicematrix / Hbrace }
9593 {
9594   color .code:n = ,
9595   horizontal-label .code:n = ,
9596   horizontal-labels .code:n = ,
9597   shorten .code:n = ,
9598   shorten-start .code:n = ,
9599   shorten-end .code:n = ,
9600   shorten+ .code:n = ,
9601   shorten-start+ .code:n = ,
9602   shorten-end+ .code:n = ,
9603   shorten~+ .code:n = ,
9604   shorten-start~+ .code:n = ,
9605   shorten-end~+ .code:n = ,
9606   brace-shift .code:n = ,
9607   brace-shift+ .code:n = ,
9608   brace-shift~+ .code:n = ,
9609   unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9610 }

```

Here we need an “fully expandable” command.

```

9611 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }

```

```

9612 {
9613   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9614     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9615     { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9616 }

```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

9617 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9618 {
9619   \int_compare:nNnTF \c@iRow < { 2 }
9620   {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9621     \str_if_eq:nnTF { #2 } { * }
9622     {
9623       \bool_set_true:N \l_@@_nullify_dots_bool
9624       \Ldots
9625       [
9626         line-style = nicematrix / brace ,
9627         #1 ,
9628         up =
9629         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9630       ]
9631     }
9632     {
9633       \Hdotsfor
9634       [
9635         line-style = nicematrix / brace ,
9636         #1 ,
9637         up =
9638         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9639       ]
9640       { #2 }
9641     }
9642   }
9643   {
9644     \str_if_eq:nnTF { #2 } { * }
9645     {
9646       \bool_set_true:N \l_@@_nullify_dots_bool
9647       \Ldots
9648       [
9649         line-style = nicematrix / mirrored-brace ,
9650         #1 ,
9651         down =
9652         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9653       ]
9654     }
9655     {
9656       \Hdotsfor
9657       [
9658         line-style = nicematrix / mirrored-brace ,
9659         #1 ,
9660         down =
9661         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9662       ]
9663       { #2 }
9664     }
9665   }
9666   \keys_set:nn { nicematrix / Hbrace } { #1 }
9667 }

```

```

9668 \NewDocumentCommand { \@@_Vbrace } { 0 { } m m }

```



```

9669 {
9670   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9671   { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9672   { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9673 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

9674 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
9675 {
9676   \int_compare:nNnTF \c@jCol < { 2 }
9677   {
9678     \str_if_eq:nnTF { #2 } { * }
9679     {
9680       \bool_set_true:N \l_@@_nullify_dots_bool
9681       \Vdots
9682       [
9683         Vbrace ,
9684         line-style = nicematrix / mirrored-brace ,
9685         #1 ,
9686         down =
9687         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9688       ]
9689     }
9690     {
9691       \Vdotsfor
9692       [
9693         Vbrace ,
9694         line-style = nicematrix / mirrored-brace ,
9695         #1 ,
9696         down =
9697         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9698       ]
9699       { #2 }
9700     }
9701   }
9702   {
9703     \str_if_eq:nnTF { #2 } { * }
9704     {
9705       \bool_set_true:N \l_@@_nullify_dots_bool
9706       \Vdots
9707       [
9708         Vbrace ,
9709         line-style = nicematrix / brace ,
9710         #1 ,
9711         up =
9712         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9713       ]
9714     }
9715     {
9716       \Vdotsfor
9717       [
9718         Vbrace ,
9719         line-style = nicematrix / brace ,
9720         #1 ,
9721         up =
9722         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9723       ]
9724       { #2 }
9725     }
9726   }
9727   \keys_set:nn { nicematrix / Hbrace } { #1 }
9728 }

```

35 The command TikzEveryCell

```

9729 \bool_new:N \l_@@_not_empty_bool
9730 \bool_new:N \l_@@_empty_bool
9731
9732 \keys_define:nn { nicematrix / TikzEveryCell }
9733 {
9734   not-empty .code:n =
9735     \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
9736     { \bool_set_true:N \l_@@_not_empty_bool }
9737     { \@@_error:n { detection-of-empty-cells } } ,
9738   not-empty .value_forbidden:n = true ,
9739   empty .code:n =
9740     \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
9741     { \bool_set_true:N \l_@@_empty_bool }
9742     { \@@_error:n { detection-of-empty-cells } } ,
9743   empty .value_forbidden:n = true ,
9744   unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
9745 }
9746
9747
9748 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9749 {
9750   \IfPackageLoadedTF { tikz }
9751   {
9752     \group_begin:
9753     \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.

```

9754     \tl_set:Nn \l_tmpa_tl { { #2 } }
9755     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9756     { \@@_for_a_block:nnnnn ##1 }
9757     \@@_all_the_cells:
9758     \group_end:
9759   }
9760   { \@@_error:n { TikzEveryCell-without-tikz } }
9761 }
9762
9763
9764 \cs_new_protected:Nn \@@_all_the_cells:
9765 {
9766   \int_step_inline:nn \c@iRow
9767   {
9768     \int_step_inline:nn \c@jCol
9769     {
9770       \cs_if_exist:cF { cell - ##1 - #####1 }
9771       {
9772         \clist_if_in:NcF \l_@@_corners_cells_clist
9773         { ##1 - #####1 }
9774         {
9775           \bool_set_false:N \l_tmpa_bool
9776           \cs_if_exist:cTF
9777           { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
9778           {
9779             \bool_if:NF \l_@@_empty_bool
9780             { \bool_set_true:N \l_tmpa_bool }
9781           }
9782           {
9783             \bool_if:NF \l_@@_not_empty_bool
9784             { \bool_set_true:N \l_tmpa_bool }
9785           }
9786         }

```

```

9787         {
9788             \@@_block_tikz:onnenn
9789             \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
9790         }
9791     }
9792 }
9793 }
9794 }
9795 }
9796
9797 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9798 {
9799     \bool_if:NF \l_@@_empty_bool
9800     {
9801         \@@_block_tikz:onnenn
9802         \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9803     }
9804     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9805 }
9806
9807 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9808 {
9809     \int_step_inline:nnn { #1 } { #3 }
9810     {
9811         \int_step_inline:nnn { #2 } { #4 }
9812         { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9813     }
9814 }

```

36 The command \CreateBlocksInColumn

```

9815 \cs_new_protected:Npn \@@_create_blocks_in_col:n #1
9816 { \clist_gput_right:Nn \g_@@_cbic_clist { #1 } }
9817
9818 \cs_new_protected:Npn \@@_cbic:
9819 {
9820     \clist_map_inline:Nn \g_@@_cbic_clist
9821     {
9822         \cs_set:cpn
9823         {
9824             pgf @ sh @ ns @ \@@_env:
9825             - \int_eval:n { \c@iRow + 1 } - ##1 }
9826         { rien }
9827     }
9828 }
9829
9830 \l_tmpa_int will be the first row of the block being created.
9831
9832 \int_set:Nn \l_tmpa_int { 1 }
9833 \int_step_inline:nn { \c@iRow + 1 }
9834 {
9835     \cs_if_exist:cT
9836     { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 }
9837     {
9838         \int_compare:nNnT { #####1 } > { \l_tmpa_int + 1 }
9839         {
9840             \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
9841             {
9842                 { \int_use:N \l_tmpa_int }
9843                 { ##1 }
9844                 { \int_eval:n { #####1 - 1 } }
9845                 { ##1 }
9846                 { }
9847             }
9848         }
9849     }
9850     \int_set:Nn \l_tmpa_int { #####1 }
9851 }

```

```

9844     }
9845   }
9846 }
9847 \clist_gclear:N \g_@@_cbic_clist
9848 }

```

37 The command \ShowCellNames

```

9849 \NewDocumentCommand \@@_ShowCellNames { }
9850 {
9851   \bool_if:NT \l_@@_in_code_after_bool
9852   {
9853     \pgfpicture
9854     \pgfrememberpicturepositiononpagetrue
9855     \pgf@relevantforpicturesizefalse
9856     \pgfpathrectanglecorners
9857     { \@@_qpoint:n { 1 } }
9858     { \@@_qpoint:n { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } } }
9859     \pgfsetfillopacity { 0.75 }
9860     \pgfsetfillcolor { white }
9861     \pgfusepathqfill
9862     \endpgfpicture
9863   }
9864   \dim_gzero_new:N \g_@@_tmpc_dim
9865   \dim_gzero_new:N \g_@@_tmpd_dim
9866   \dim_gzero_new:N \g_@@_tmpe_dim
9867   \int_step_inline:nn \c@iRow
9868   {
9869     \bool_if:NTF \l_@@_in_code_after_bool
9870     {
9871       \pgfpicture
9872       \pgfrememberpicturepositiononpagetrue
9873       \pgf@relevantforpicturesizefalse
9874     }
9875     { \begin { pgfpicture } }
9876     \@@_qpoint:n { row - ##1 }
9877     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9878     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9879     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9880     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9881     \bool_if:NTF \l_@@_in_code_after_bool
9882     { \endpgfpicture }
9883     { \end { pgfpicture } }
9884     \int_step_inline:nn \c@jCol
9885     {
9886       \hbox_set:Nn \l_tmpa_box
9887       {
9888         \normalfont \Large \sffamily \bfseries
9889         \bool_if:NTF \l_@@_in_code_after_bool
9890         { \color { red } }
9891         { \color { red ! 50 } }
9892         ##1 - ####1
9893       }
9894       \bool_if:NTF \l_@@_in_code_after_bool
9895       {
9896         \pgfpicture
9897         \pgfrememberpicturepositiononpagetrue
9898         \pgf@relevantforpicturesizefalse
9899       }
9900       { \begin { pgfpicture } }
9901       \@@_qpoint:n { col - ####1 }
9902       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x

```

```

9903 \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9904 \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9905 \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9906 \bool_if:NTF \l_@@_in_code_after_bool
9907 { \endpgfpicture }
9908 { \end { pgfpicture } }
9909 \fp_set:Nn \l_tmpa_fp
9910 {
9911 \fp_min:nn
9912 {
9913 \fp_min:nn
9914 { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9915 { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9916 }
9917 { 1.0 }
9918 }
9919 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9920 \pgfpicture
9921 \pgfrememberpicturepositiononpagetrue
9922 \pgf@relevantforpicturesizefalse
9923 \pgftransformshift
9924 {
9925 \pgfpoint
9926 { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9927 \g_tmpa_dim
9928 }
9929 \pgfnode
9930 { rectangle }
9931 { center }
9932 { \box_use:N \l_tmpa_box }
9933 { }
9934 { }
9935 \endpgfpicture
9936 }
9937 }
9938 }

```

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.

```

9939 \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.

```

9940 \bool_new:N \g_@@_footnote_bool
9941 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9942 {
9943   You~have~used~the~key~' \l_keys_key_str '~when~loading~nicematrix~
9944   but~that~key~is~unknown. \\\
9945   It~will~be~ignored. \\\
9946   For~a~list~of~the~available~keys,~type~H~<return>.
9947 }
9948 {
9949   The~available~keys~are~(in~alphabetic~order):~
9950   footnote,~

```

```

9951     footnotehyper,~
9952     messages-for-Overleaf,~
9953     renew-dots~and~
9954     renew-matrix.
9955 }
9956 \keys_define:nn { nicematrix }
9957 {
9958     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9959     renew-dots .value_forbidden:n = true ,
9960     renew-matrix .code:n = \@@_renew_matrix: ,
9961     renew-matrix .value_forbidden:n = true ,
9962     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9963     footnote .bool_set:N = \g_@@_footnote_bool ,
9964     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9965     unknown .code:n = \@@_error:n { Unknown~key~for~package }
9966 }
9967 \ProcessKeyOptions

9968 \@@_msg_new:nn { footnote-with-footnotehyper-package }
9969 {
9970     You~can't~use~the~option~'footnote'~because~the~package~
9971     footnotehyper~has~already~been~loaded.~
9972     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9973     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9974     of~the~package~footnotehyper.\\
9975     The~package~footnote~won't~be~loaded.
9976 }

9977 \@@_msg_new:nn { footnotehyper-with-footnote-package }
9978 {
9979     You~can't~use~the~option~'footnotehyper'~because~the~package~
9980     footnote~has~already~been~loaded.~
9981     If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9982     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9983     of~the~package~footnote.\\
9984     The~package~footnotehyper~won't~be~loaded.
9985 }

9986 \bool_if:NT \g_@@_footnote_bool
9987 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9988     \IfClassLoadedTF { beamer }
9989     { \bool_set_false:N \g_@@_footnote_bool }
9990     {
9991         \IfPackageLoadedTF { footnotehyper }
9992         { \@@_error:n { footnote-with-footnotehyper-package } }
9993         { \usepackage { footnote } }
9994     }
9995 }

9996 \bool_if:NT \g_@@_footnotehyper_bool
9997 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9998     \IfClassLoadedTF { beamer }
9999     { \bool_set_false:N \g_@@_footnote_bool }
10000     {
10001         \IfPackageLoadedTF { footnote }
10002         { \@@_error:n { footnotehyper-with-footnote-package } }
10003         { \usepackage { footnotehyper } }

```

```

10004     }
10005     \bool_set_true:N \g_@@_footnote_bool
10006 }

```

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.

```

10007 \bool_new:N \l_@@_underscore_loaded_bool
10008 \IfPackageLoadedT { underscore }
10009 { \bool_set_true:N \l_@@_underscore_loaded_bool }
10010 \hook_gput_code:nnn { begindocument } { . }
10011 {
10012     \bool_if:NF \l_@@_underscore_loaded_bool
10013     {
10014         \IfPackageLoadedT { underscore }
10015         { \@@_error:n { underscore~after~nicematrix } }
10016     }
10017 }

```

40 Compatibility with threeparttable

```

10018 \hook_gput_code:nnn { begindocument } { . }
10019 {
10020     \IfPackageLoadedT { threeparttable }
10021     {
10022         \AddToHook { env / threeparttable / begin }
10023         {
10024             \TPT@hookin { NiceTabular }
10025             \TPT@hookin { NiceTabular* }
10026             \TPT@hookin { NiceTabularX }
10027         }
10028     }
10029 }

```

41 Error messages of the package

When there is a unknown key, maybe the user has tried to use an inexistent “additive syntax” for that key. Of course, in that case, the last character of the name of the key is `+`.

`#1` is a clist of names of sets of keys and `#2` is the error message to send.

```

10030 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
10031 {
10032     \str_if_eq:eeTF
10033     { \str_item:Nn \l_keys_key_str { \str_count:N \l_keys_key_str } }
10034     { + }
10035     {
10036         \str_set:Ne \l_tmpa_str
10037         { \str_range:Nnn \l_keys_key_str { 1 } { \str_count:N \l_keys_key_str - 1 } }

```

```

10038     \bool_set_false:N \l_tmpa_bool
10039     \clist_map_inline:nn { #1 }
10040     {
10041         \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10042         {
10043             \@@_error:n { key~without~+~exists }
10044             \bool_set_true:N \l_tmpa_bool
10045             \clist_map_break:
10046         }
10047     }
10048     \bool_if:NF \l_tmpa_bool
10049     {
10050         \str_set:Ne \l_keys_key_str { \tl_trim_right_spaces:V \l_tmpa_str }
10051         \@@_unknown_key_i:nn { #1 } { #2 }
10052     }
10053 }
10054 { \@@_unknown_key_i:nn { #1 } { #2 } }
10055 }

```

We try a normalisation of the name 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.

```

10056 \cs_new_protected:Npn \@@_unknown_key_i:nn #1 #2
10057 {
10058     \str_set_eq:NN \l_tmpa_str \l_keys_key_str
10059     \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
10060     \str_set:Ne \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
10061     \bool_set_false:N \l_tmpa_bool
10062     \clist_map_inline:nn { #1 }
10063     {
10064         \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10065         {
10066             \@@_error:n { key~with~normal~form~exists }
10067             \bool_set_true:N \l_tmpa_bool
10068             \clist_map_break:
10069         }
10070     }
10071     \bool_if:NF \l_tmpa_bool
10072     {
10073         \@@_error:n { #2 }

```

If `messages-for-Overleaf` is not in force, the list of the available keys is not written in the main error message but only in the complement (if the final user presses H). That's why we write, in all circumstances, the list of the available keys in order to facilitate the work of the systems which analyze the error by IA (such as Prism).

```

10074     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10075     { \msg_info:nn { nicematrix } { #2~+ } }
10076 }
10077 }
10078 \@@_msg_new:nn { key~without~+~exists }
10079 {
10080     The~key~'\tl_trim_right_spaces:V \l_tmpa_str'~exists~but~does~not~accept~an~
10081     additive~syntax~(with~+=).\
10082     It~will~be~ignored.\
10083 }
10084 \@@_msg_new:nn { key~with~normal~form~exists }
10085 {
10086     No~key~'\l_keys_key_str'.\
10087     It~will~be~ignored.\
10088     Maybe~you~want~to~use~the~key~'\l_tmpa_str'.

```



```

10089 }
10090 \str_const:Ne \c_@@_available_keys_str
10091 {
10092   \bool_if:nT { ! \g_@@_messages_for_Overleaf_bool }
10093   { For~a~list~of~the~available~keys,~type~H~<return>. }
10094 }
10095 \seq_new:N \g_@@_types_of_matrix_seq
10096 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
10097 {
10098   NiceMatrix ,
10099   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
10100 }
10101 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
10102 { \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`.

```

10103 \cs_new_protected:Npn \@@_err_too_many_cols:
10104 {
10105   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
10106   { \@@_fatal:nn { too-many-cols-for-array } }
10107   \int_compare:nNnT \l_@@_last_col_int = { -2 }
10108   { \@@_fatal:n { too-many-cols-for-matrix } }
10109   \int_compare:nNnT \l_@@_last_col_int = { -1 }
10110   { \@@_fatal:n { too-many-cols-for-matrix } }
10111   \bool_if:NF \l_@@_last_col_without_value_bool
10112   { \@@_fatal:n { too-many-cols-for-matrix-with-last-col } }
10113 }

```

The following command must *not* be protected since it's used in an error message.

```

10114 \cs_new:Npn \@@_message_hdotsfor:
10115 {
10116   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
10117   { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
10118     \token_to_str:N \Hbrace \ is~incorrect. }
10119 }
10120 \cs_new_protected:Npn \@@_Hline_in_cell:
10121 { \@@_fatal:n { Misuse-of-Hline } }
10122 \@@_msg_new:nn { Misuse-of-Hline }
10123 {
10124   Misuse~of~Hline. \\
10125   Error~in~your~row~ \int_eval:N \c@iRow . \\
10126   \token_to_str:N \Hline\ (like \token_to_str:N \hline)~must~be~used~only~
10127   at~the~beginning~of~a~row.\\
10128   That~error~is~fatal.
10129 }
10130 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
10131 {
10132   Incompatible~options.\\
10133   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~the~same~time.\\
10134   The~output~will~not~be~reliable.
10135 }
10136 \@@_msg_new:nn { Body-alone }
10137 {
10138   \token_to_str:N \Body\ alone. \\
10139   You~have~used~\token_to_str:N \Body\ without~\token_to_str:N \CodeBefore.\\
10140   That~error~is~fatal.
10141 }

```

```

10142 \@@_msg_new:nn { NiceTabularX~probably~required }
10143 {
10144   Incorrect~syntax.\\
10145   You~probably~want~to~use~\{NiceTabularX\}~whose~first~argument~is~the~
10146   disered~width~of~the~tabular.\\
10147   That~error~is~fatal.
10148 }
10149 \@@_msg_new:nn { cellcolor~in~Block }
10150 {
10151   Bad~use~of~\token_to_str:N \cellcolor \\
10152   You~can't~use~\token_to_str:N \cellcolor\ in~\token_to_str:N \Block\
10153   \bool_if:NTF \l_@@_amp_in_blocks_bool
10154   { (but~you~could~use~it~in~a~sub~block~since~'&~in~blocks'~is~in~force) }
10155   { (it's~possible~in~a~sub~block~when~'&~in~blocks'~is~in~force) }
10156   .~Here,~you~should~use~the~key~'fill'~of~the~block.\\
10157   That~command~will~be~ignored.
10158 }
10159 \@@_msg_new:nn { rowcolor~in~Block }
10160 {
10161   Bad~use~of~\token_to_str:N \rowcolor \\
10162   You~can't~use~\token_to_str:N \rowcolor\ in~\token_to_str:N \Block.\\
10163   However,~it's~possible~to~color~the~block~with~its~key~'fill'.\\
10164   That~command~will~be~ignored.
10165 }
10166 \@@_msg_new:nn { key~color~inside }
10167 {
10168   Deleted~key.\\
10169   The~key~'color~inside'~(and~its~alias~'colortbl~like')~has~been~deleted~in
10170   ~'nicematrix'~and~must~not~be~used.\\
10171   This~error~is~fatal.
10172 }
10173 \@@_msg_new:nn { Invalid~argument~for~w }
10174 {
10175   Invalid~argument~for~w.\\
10176   You~have~used~the~type~of~alignment~'#1'~for~your~column~'w'~
10177   but~only~'c',~'r',~'l'~and~'s'~are~allowed~in~a~column~'w'.~
10178   If~you~go~on,~'c'~will~be~used.
10179 }
10180 \@@_msg_new:nn { invalid~weight }
10181 {
10182   Unknown~key.\\
10183   The~key~'\l_keys_key_str'~of~your~column~X~is~unknown~and~will~be~ignored.~
10184   The~available~keys~are:~l,~c,~r,~t~(=p),~m,~b,~V~
10185   \IfPackageLoadedTF { varwidth }
10186   { (since~'varwidth'~is~loaded)~}
10187   { (if~you~load~'varwidth')~}
10188   and~real~numbers~for~the~weight~of~the~X~column.
10189 }
10190 \@@_msg_new:nn { last~col~not~used }
10191 {
10192   Column~not~used.\\
10193   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
10194   in~your~\@@_full_name_env: .~
10195   However,~you~can~go~on.
10196 }
10197 \@@_msg_new:nn { too~many~cols~for~matrix~with~last~col }
10198 {
10199   Too~many~columns.\\
10200   In~the~row~\int_eval:n { \c@iRow },~
10201   you~try~to~use~more~columns~
10202   than~allowed~by~your~\@@_full_name_env: .

```

```

10203 \@@_message_hdotsfor: \
10204 The~maximal~number~of~columns~is~ \int_eval:n { \l_@@_last_col_int - 1 }~
10205 (plus~the~exterior~columns).\\
10206 But,~maybe,~you~have~forgotten~a~\token_to_str:N \\. \\
10207 This~error~is~fatal.
10208 }
10209 \@@_msg_new:nn { too-many-cols-for-matrix }
10210 {
10211 Too-many-columns.\\
10212 In~the~row~ \int_eval:n { \c@iRow } ,~
10213 you~try~to~use~more~columns~than~allowed~by~your~ \@@_full_name_env: .
10214 \@@_message_hdotsfor: \
10215 Recall~that~the~maximal~number~of~columns~for~a~matrix~
10216 (excepted~the~potential~exterior~columns)~is~fixed~by~the~
10217 LaTeX~counter~'MaxMatrixCols'.~
10218 Its~current~value~is~ \int_use:N \c@MaxMatrixCols \
10219 (use~ \token_to_str:N \setcounter \ to~change~that~value).\\
10220 But,~maybe,~you~have~forgotten~a~\token_to_str:N \\. \\
10221 This~error~is~fatal.
10222 }
10223 \@@_msg_new:nn { too-many-cols-for-array }
10224 {
10225 Too-many-columns.\\
10226 In~the~row~ \int_eval:n { \c@iRow } ,~
10227 ~you~try~to~use~more~columns~than~allowed~by~your~
10228 \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal~number~of~columns~is~
10229 \int_use:N \g_@@_static_num_of_col_int \
10230 \bool_if:nT
10231 { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
10232 { (plus~the~exterior~ones)~}
10233 since~the~preamble~is~' \g_@@_user_preamble_tl '.\\
10234 But,~maybe,~you~have~forgotten~a~\token_to_str:N \\. \\
10235 This~error~is~fatal.
10236 }
10237 \@@_msg_new:nn { columns-not-used }
10238 {
10239 Columns-not-used.\\
10240 The~preamble~of~your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '~
10241 It~announces~ \int_use:N \g_@@_static_num_of_col_int \
10242 columns~but~you~only~used~ \int_use:N \c@jCol .\\
10243 The~columns~you~did~not~used~won't~be~created.\\
10244 You~won't~have~similar~warning~till~the~end~of~the~document.
10245 }
10246 }
10247 \@@_msg_new:nn { Bad-use-of-NiceTabularNotes }
10248 {
10249 Bad-use-of~\token_to_str:N \NiceTabularNotes \\
10250 \token_to_str:N~\NiceTabularNotes\ should~be~used~only~
10251 after~at~tabular~which~uses~`notes/no-print`. \\
10252 That~command~will~be~ignored.
10253 }
10254 \@@_msg_new:nn { empty-preamble }
10255 {
10256 Empty~preamble.\\
10257 The~preamble~of~your~ \@@_full_name_env: \ is~empty.\\
10258 This~error~is~fatal.
10259 }
10260 \@@_msg_new:nn { in-first-col }
10261 {
10262 Erroneous~use.\\

```

```

10263     You-can't-use-the-command-#1 in-the-first-column-(number-0)-of-the-array.\\
10264     That-command-will-be-ignored.\\
10265     You-can-try-to-delete-the-key-'first-col'.
10266 }

10267 \@@_msg_new:nn { in~last~col }
10268 {
10269     Erroneous-use.\\
10270     You-can't-use-the-command-#1 in-the-last-column-(exterior)-of-the-array.\\
10271     That-command-will-be-ignored.\\
10272     You-can-try-to-delete-the-key-'last-col'.
10273 }

10274 \@@_msg_new:nn { in~first~row }
10275 {
10276     Erroneous-use.\\
10277     You-can't-use-the-command-#1 in-the-first-row-(number-0)-of-the-array.\\
10278     That-command-will-be-ignored.\\
10279     You-can-try-to-delete-the-key-'first-row'.
10280 }

10281 \@@_msg_new:nn { in~last~row }
10282 {
10283     Erroneous-use.\\
10284     You-can't-use-the-command-#1 in-the-last-row-(exterior)-of-the-array.\\
10285     That-command-will-be-ignored.\\
10286     You-can-try-to-delete-the-key-'last-row'.
10287 }

10288 \@@_msg_new:nn { TopRule~without~booktabs }
10289 {
10290     Erroneous-use.\\
10291     You-can't-use-the-command- #1 because-'booktabs'~is-not-loaded.\\
10292     You-should-load-'booktabs'~(before~or~after~'nicematrix').\\
10293     That-command-will-be-ignored.
10294 }

10295 \@@_msg_new:nn{ rotate~in~p~col }
10296 {
10297     \token_to_str:N \rotate\ forbidden.\\
10298     You-should-not-use-\token_to_str:N \rotate\ in-a-column-of-type~'p',~
10299     'b',~'m'\IfPackageLoadedTF { varwidth } { ,~'X'~or~'V' } { ~or~'X'}.~
10300     If-you-go-on,~maybe-you-won't-have-the-expected-output.
10301 }

10302 \@@_msg_new:nn { TopRule~without~tikz }
10303 {
10304     Erroneous-use.\\
10305     You-can't-use-the-command- #1 because-TikZ-is-not-loaded.\\
10306     You-should-load-TikZ-with-\token_to_str:N \usepackage \{tikz\}.\\
10307     \IfPackageLoadedF { booktabs }
10308     { You-should-also-load-'booktabs'.\\ }
10309     That-command-will-be-ignored.
10310 }

10311 \@@_msg_new:nn { caption~outside~float }
10312 {
10313     Key-caption-forbidden.\\
10314     You-can't-use-the-key~'caption'~because-you-are-not-in-a-floating-
10315     environment~(such-as~\{table\}).~This-key-will-be-ignored.
10316 }

10317 \@@_msg_new:nn { short-caption~without~caption }
10318 {
10319     You-should-not-use-the-key~'short-caption'~without~'caption'.~
10320     However,~your~'short-caption'~will-be-used-as~'caption'.
10321 }

10322 \@@_msg_new:nn { double~closing~delimiter }

```

```

10323 {
10324   Double~delimiter.\\
10325   You-can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
10326   delimiter.~This~delimiter~will~be~ignored.\\
10327   You-can~try~to~use~\token_to_str:N \SubMatrix\ in~the~\token_to_str:N \CodeAfter.
10328 }
10329 \@@_msg_new:nn { delimiter~after~opening }
10330 {
10331   Double~delimiter.\\
10332   You-can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
10333   delimiter.~That~delimiter~will~be~ignored.
10334 }
10335 \@@_msg_new:nn { bad~option~for~line~style }
10336 {
10337   Bad~line~style.\\
10338   Since~you~haven't~loaded~TikZ,~the~only~value~you~can~give~to~'line~style'~
10339   is~'standard'.~That~key~will~be~ignored.\\
10340   You-can~load~TikZ~with~\token_to_str:N \usepackage \{tikz\},~
10341   before~or~after~'nicematrix'.\\
10342 }
10343 \@@_msg_new:nn { corners~with~no~cell~nodes }
10344 {
10345   Incompatible~keys.\\
10346   You-can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
10347   is~in~force~(you~should~deactive~the~key~'no~cell~nodes'~whose~only~goal~
10348   is~to~speed~up~compilation).\\
10349   If~you~go~on,~that~key~will~be~ignored.
10350 }
10351 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
10352 {
10353   Incompatible~keys.\\
10354   You-can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
10355   is~in~force~(you~should~deactive~the~key~'no~cell~nodes'~whose~only~goal~
10356   is~to~speed~up~compilation).\\
10357   If~you~go~on,~those~extra~nodes~won't~be~created.
10358 }
10359 \@@_msg_new:nn { Identical~notes~in~caption }
10360 {
10361   Identical~tabular~notes.\\
10362   You-can't~put~several~notes~with~the~same~content~in~
10363   \token_to_str:N \caption \ (but~it's~possible~in~the~main~tabular).\\
10364   If~you~go~on,~the~output~will~probably~be~erroneous.
10365 }
10366 \@@_msg_new:nn { tabularnote~below~the~tabular }
10367 {
10368   \token_to_str:N \tabularnote \ forbidden\\
10369   You-can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
10370   of~your~tabular~because~the~caption~will~be~composed~below~
10371   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
10372   key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .\\
10373   Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
10374   no~similar~error~will~raised~in~this~document.
10375 }
10376 \@@_msg_new:nn { Unknown~key~for~rules }
10377 {
10378   Unknown~key.\\
10379   There~is~only~two~keys~available~here:~width~and~color.\\
10380   Your~key~' \l_keys_key_str ' ~will~be~ignored.
10381 }
10382 \@@_msg_new:nn { Unknown~key~for~Hbrace }

```

```

10383 {
10384   Unknown~key.\\
10385   You~have~used~the~key~' \l_keys_key_str '~but~the~only~
10386   keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
10387   and~ \token_to_str:N \Vbrace \ are::~'brace-shift(+)',~'color',~
10388   'horizontal-label(s)',~'shorten'~'shorten-end'~
10389   and~'shorten-start'.\\
10390   That~error~is~fatal.
10391 }
10392 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
10393 {
10394   Unknown~key.\\
10395   There~is~only~two~keys~available~here::~
10396   'empty'~and~'not-empty'.\\
10397   Your~key~' \l_keys_key_str '~will~be~ignored.
10398 }
10399 \@@_msg_new:nn { Unknown~key~for~rotate }
10400 {
10401   Unknown~key.\\
10402   The~only~key~available~here~is~'c'.\\
10403   Your~key~' \l_keys_key_str '~will~be~ignored.
10404 }
10405 \@@_msg_new:nnn { Unknown~key~for~custom~line }
10406 {
10407   Unknown~key.\\
10408   The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
10409   It~you~go~on,~you~will~probably~have~other~errors. \\
10410   \c_@@_available_keys_str
10411 }
10412 {
10413   The~available~keys~are~(in~alphabetic~order)::~
10414   ccommand,~
10415   color,~
10416   command,~
10417   dotted,~
10418   letter,~
10419   multiplicity,~
10420   sep-color,~
10421   tikz,~and~total-width.
10422 }
10423 \@@_msg_new:nnn { Unknown~key~for~xdots }
10424 {
10425   Unknown~key.\\
10426   The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
10427   \c_@@_available_keys_str
10428 }
10429 {
10430   The~available~keys~are~(in~alphabetic~order)::~
10431   'color',~
10432   'horizontal(s)-labels',~
10433   'inter',~
10434   'line-style',~
10435   'nullify',~
10436   'radius',~
10437   'shorten',~
10438   'shorten-end'~and~'shorten-start'.
10439 }
10440 \@@_msg_new:nn { Unknown~key~for~rowcolors }
10441 {
10442   Unknown~key.\\
10443   As~for~now,~there~is~only~two~keys~available~here::~'cols'~and~'respect-blocks'~
10444   (and~you~try~to~use~' \l_keys_key_str '~)\\

```

```

10445     That~key~will~be~ignored.
10446 }
10447 \@@_msg_new:nn { label~without~caption }
10448 {
10449     You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
10450     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
10451 }
10452 \@@_msg_new:nn { W~warning }
10453 {
10454     Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
10455     (row~ \int_use:N \c@iRow ).
10456 }
10457 \@@_msg_new:nn { Construct~too~large }
10458 {
10459     Construct~too~large.\\
10460     Your~command~ \token_to_str:N #1
10461     can't~be~drawn~because~your~matrix~is~too~small.\\
10462     That~command~will~be~ignored.
10463 }
10464 \@@_msg_new:nn { underscore~after~nicematrix }
10465 {
10466     Problem~with~'underscore'.\\
10467     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
10468     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
10469     ' \token_to_str:N \Cdots \token_to_str:N _
10470     \{ n \token_to_str:N \text \{ ~times \} \}' .
10471 }
10472 \@@_msg_new:nn { ampersand~in~light-syntax }
10473 {
10474     Ampersand~forbidden.\\
10475     You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
10476     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
10477 }
10478 \@@_msg_new:nn { double~backslash~in~light-syntax }
10479 {
10480     Double~backslash~forbidden.\\
10481     You~can't~use~ \token_to_str:N \\
10482     ~to~separate~rows~because~the~key~'light-syntax'~
10483     is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
10484     (set~by~the~key~'end-of-row').~This~error~is~fatal.
10485 }
10486 \@@_msg_new:nn { hlines~with~color }
10487 {
10488     Incompatible~keys.\\
10489     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
10490     \token_to_str:N \Block \ when~the~key~'color'~or~'draw'~is~used.\\
10491     However,~you~can~put~several~commands~ \token_to_str:N \Block.\\
10492     Your~key~will~be~discarded.
10493 }
10494 \@@_msg_new:nn { bad~value~for~baseline }
10495 {
10496     Bad~value~for~baseline.\\
10497     The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
10498     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
10499     \int_use:N \g_@@_row_total_int \ or~equal~to~'t',~'c'~or~'b'~or~of~
10500     the~form~'line-i'.\\
10501     A~value~of~1~will~be~used.
10502 }
10503 \@@_msg_new:nn { bad~value~for~baseline~line }
10504 {

```

```

10505     Bad-value-for-baseline-with-line.\\
10506     The-value-given-to-'baseline'~( \int_use:N \l_tmpa_int )~is-not~
10507     valid.~The-number-of-the-line-must-be-between~1-and~
10508     \int_eval:n { \c@iRow + 1 } \\
10509     A-value-of-'line-1'~will-be-used.
10510 }
10511 \@@_msg_new:nn { detection-of-empty-cells }
10512 {
10513     Problem-with-'not-empty'\\
10514     For-technical-reasons,~you-must-activate~
10515     'create-cell-nodes'~in~ \token_to_str:N \CodeBefore \
10516     in-order-to-use-the-key~' \l_keys_key_str '.\\
10517     That-key-will-be-ignored.
10518 }
10519 \@@_msg_new:nn { siunitx-not-loaded }
10520 {
10521     siunitx-not-loaded\\
10522     You-can't-use-the-columns-'S'~because-'siunitx'~is-not-loaded.\\
10523     That-error-is-fatal.\\
10524     You-can-load-'siunitx'~with~\token_to_str:N \usepackage \{siunitx\},~
10525     before-or-after-'nicematrix'. \\
10526 }
10527 \@@_msg_new:nn { Invalid-name }
10528 {
10529     Invalid-name.\\
10530     You-can't-give-the-name~' \l_keys_value_tl '~to-a~ \token_to_str:N
10531     \SubMatrix \ of~your~ \@@_full_name_env: .\\
10532     A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\\
10533     This-key-will-be-ignored.
10534 }
10535 \@@_msg_new:nn { Hbrace-not-allowed }
10536 {
10537     Command-not-allowed.\\
10538     You-can't-use-the-command~ \token_to_str:N #1
10539     because-you-have-not-loaded~
10540     \IfPackageLoadedTF { tikz }
10541     { the-TikZ-library~'decorations.pathreplacing'~Use~ }
10542     { TikZ~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
10543     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\
10544     That-command-will-be-ignored.
10545 }
10546 \@@_msg_new:nn { Vbrace-not-allowed }
10547 {
10548     Command-not-allowed.\\
10549     You-can't-use-the-command~ \token_to_str:N \Vbrace \
10550     because-you-have-not-loaded-TikZ~
10551     and-the-TikZ-library~'decorations.pathreplacing'.\\
10552     Use: ~\token_to_str:N \usepackage \{tikz\}~
10553     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
10554     That-command-will-be-ignored.
10555 }
10556 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
10557 {
10558     Wrong-line.\\
10559     You-try-to-draw-a-#1~line-of-number~'#2'~in-a~
10560     \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
10561     number-is-not-valid.~It-will-be-ignored.
10562 }
10563 \@@_msg_new:nn { Impossible-delimiter }
10564 {
10565     Impossible-delimiter.\\

```



```

10566     It's~impossible~to~draw~the~#1~delimiter~of~your~
10567     \token_to_str:N \SubMatrix \ because~all~the~cells~are~empty~
10568     in~that~column.
10569     \bool_if:NT \l_@@_submatrix_slim_bool
10570     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
10571     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
10572 }
10573 \@@_msg_new:nnn { width~without~X~columns }
10574 {
10575     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
10576     the~preamble~(' \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .\\
10577     That~key~will~be~ignored.
10578 }
10579 {
10580     This~message~is~the~message~'width~without~X~columns'~
10581     of~the~module~'nicematrix'.~
10582     The~experimented~users~can~disable~that~message~with~
10583     \token_to_str:N \msg_redirect_name:nnn .\\
10584 }
10585
10586 \@@_msg_new:nn { key~multiplicity~with~dotted }
10587 {
10588     Incompatible~keys. \\
10589     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
10590     in~a~'custom~line'.~They~are~incompatible. \\
10591     The~key~'multiplicity'~will~be~discarded.
10592 }
10593 \@@_msg_new:nn { empty~environment }
10594 {
10595     Empty~environment.\\
10596     Your~ \@@_full_name_env: \ is~empty.~This~error~is~fatal.
10597 }
10598 \@@_msg_new:nn { No~letter~and~no~command }
10599 {
10600     Erroneous~use.\\
10601     Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
10602     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10603     '~'ccommand'~(to~draw~horizontal~rules).\\
10604     However,~you~can~go~on.
10605 }
10606 \@@_msg_new:nn { Forbidden~letter }
10607 {
10608     Forbidden~letter.\\
10609     You~can't~use~the~letter~'#1'~for~a~customized~line.~
10610     It~will~be~ignored.\\
10611     The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10612 }
10613 \@@_msg_new:nn { Several~letters }
10614 {
10615     Wrong~name.\\
10616     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10617     have~used~' \l_@@_letter_str ').\\
10618     It~will~be~ignored.
10619 }
10620 \@@_msg_new:nn { Delimiter~with~small }
10621 {
10622     Delimiter~forbidden.\\
10623     You~can't~put~a~delimiter~in~the~preamble~of~your~
10624     \@@_full_name_env: \
10625     because~the~key~'small'~is~in~force.\\
10626     This~error~is~fatal.

```

```

10627 }
10628 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10629 {
10630   Unknown~cell.\\
10631   Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10632   the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10633   can't~be~executed~because~a~cell~doesn't~exist.\\
10634   This~command~ \token_to_str:N \line \ will~be~ignored.
10635 }
10636 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10637 {
10638   Duplicate~name.\\
10639   The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10640   in~this~ \@@_full_name_env: .\\
10641   This~key~will~be~ignored.\\
10642   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10643   { For~a~list~of~the~names~already~used,~type~H~<return>. }
10644 }
10645 {
10646   The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
10647   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10648 }
10649 \@@_msg_new:nn { r~or~l~with~preamble }
10650 {
10651   Erroneous~use.\\
10652   You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
10653   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10654   your~ \@@_full_name_env: .\\
10655   This~key~will~be~ignored.
10656 }
10657 \@@_msg_new:nn { Hdotsfor~in~col~0 }
10658 {
10659   Erroneous~use.\\
10660   You~can't~use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\
10661   in~an~exterior~column~of~
10662   the~array.~This~error~is~fatal.
10663 }
10664 \@@_msg_new:nn { bad~corner }
10665 {
10666   Bad~corner.\\
10667   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10668   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
10669   This~specification~of~corner~will~be~ignored.
10670 }
10671 \@@_msg_new:nn { bad~border }
10672 {
10673   Bad~border.\\
10674   \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
10675   (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
10676   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10677   also~use~the~key~'tikz'
10678   \IfPackageLoadedF { tikz }
10679   { ~if~you~load~the~LaTeX~package~'tikz' } ).\\
10680   This~specification~of~border~will~be~ignored.
10681 }
10682 \@@_msg_new:nn { TikzEveryCell~without~tikz }
10683 {
10684   TikZ~not~loaded.\\
10685   You~can't~use~ \token_to_str:N \TikzEveryCell \
10686   because~you~have~not~loaded~tikz.\\
10687   You~can~load~'tikz'~with~\token_to_str:N \usepackage \{tikz\},~

```

```

10688     before~or~after~'nicematrix'. \\
10689     This~command~will~be~ignored.
10690 }

10691 \@@_msg_new:nn { tikz-key-without-tikz }
10692 {
10693     TikZ~not~loaded.\\
10694     You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N
10695     \Block '~because~you~have~not~loaded~tikz.\\
10696     You~can~load~'tikz'~with~\token_to_str:N \usepackage \{tikz\},~
10697     before~or~after~'nicematrix'. \\
10698     This~key~will~be~ignored.
10699 }

10700 \@@_msg_new:nn { Bad~argument~for~Block }
10701 {
10702     Bad~argument.\\
10703     The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
10704     be~of~the~form~'i-j'~(or~totally~empty)~and~you~have~used:~
10705     '#1'. \\
10706     If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono-cell~(as~if~
10707     the~argument~was~empty).
10708 }

10709 \@@_msg_new:nn { last-col-non-empty-for~NiceArray }
10710 {
10711     Erroneous~use.\\
10712     In~the~ \@@_full_name_env: ,~you~must~use~the~key~
10713     'last-col'~without~value.\\
10714     However,~you~can~go~on~for~this~time~
10715     (the~value~' \l_keys_value_tl '~will~be~ignored).
10716 }

10717 \@@_msg_new:nn { last-col-non-empty-for~NiceMatrixOptions }
10718 {
10719     Erroneous~use. \\
10720     In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
10721     'last-col'~without~value. \\
10722     However,~you~can~go~on~for~this~time~
10723     (the~value~' \l_keys_value_tl '~will~be~ignored).
10724 }

10725 \@@_msg_new:nn { Block~too~large~1 }
10726 {
10727     Block~too~large. \\
10728     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
10729     too~small~for~that~block. \\
10730     This~block~and~maybe~others~will~be~ignored.
10731 }

10732 \@@_msg_new:nn { Block~too~large~2 }
10733 {
10734     Block~too~large. \\
10735     The~preamble~of~your~ \@@_full_name_env: \ announces~ \int_use:N
10736     \g_@@_static_num_of_col_int \
10737     columns~but~you~use~only~ \int_use:N \c@jCol \ and~that's~why~a~block~
10738     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
10739     (&)~at~the~end~of~the~first~row~of~your~ \@@_full_name_env: . \\
10740     This~block~and~maybe~others~will~be~ignored.
10741 }

10742 \@@_msg_new:nn { unknown~column~type }
10743 {
10744     Bad~column~type. \\
10745     The~column~type~'#1'~in~your~ \@@_full_name_env: \
10746     is~unknown. \\
10747     This~error~is~fatal.
10748 }

```

```

10749 \@@_msg_new:nn { unknown~column~type~multicolumn }
10750 {
10751   Bad~column~type. \\
10752   The~column~type~'#1'~in~the~command~\token_to_str:N \multicolumn \
10753   ~of~your~ \@@_full_name_env: \
10754   is~unknown. \\
10755   This~error~is~fatal.
10756 }
10757 \@@_msg_new:nn { unknown~column~type~S }
10758 {
10759   Bad~column~type. \\
10760   The~column~type~'S'~in~your~ \@@_full_name_env: \ is~unknown. \\
10761   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10762   load~that~package. \\
10763   This~error~is~fatal.
10764 }
10765 \@@_msg_new:nn { unknown~column~type~S~multicolumn }
10766 {
10767   Bad~column~type. \\
10768   The~column~type~'S'~in~the~command~\token_to_str:N \multicolumn \
10769   of~your~ \@@_full_name_env: \ is~unknown. \\
10770   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10771   load~that~package. \\
10772   This~error~is~fatal.
10773 }
10774 \@@_msg_new:nn { tabularnote~forbidden }
10775 {
10776   Forbidden~command. \\
10777   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10778   ~here.~This~command~is~available~only~in~
10779   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
10780   the~argument~of~a~command~\token_to_str:N \caption \ included~
10781   in~an~environment~\{table\}. \\
10782   This~command~will~be~ignored.
10783 }
10784 \@@_msg_new:nn { borders~forbidden }
10785 {
10786   Forbidden~key.\\
10787   You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \
10788   because~the~option~'rounded~corners'~
10789   is~in~force~with~a~non~zero~value.\\
10790   This~key~will~be~ignored.
10791 }
10792 \@@_msg_new:nn { bottomrule~without~booktabs }
10793 {
10794   booktabs~not~loaded.\\
10795   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
10796   loaded~'booktabs'.~You~should~load~'booktabs',~before~or~
10797   after~'nicematrix'.\\
10798   This~key~will~be~ignored.
10799 }
10800 \@@_msg_new:nn { enumitem~not~loaded }
10801 {
10802   enumitem~not~loaded. \\
10803   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10804   ~because~you~haven't~loaded~'enumitem'.~We~should~load~it~
10805   (before~or~after~'nicematrix').\\
10806   All~the~commands~ \token_to_str:N \tabularnote \ will~be~
10807   ignored~in~the~document.
10808 }
10809 \@@_msg_new:nn { tikz~without~tikz }

```

```

10810 {
10811   TikZ~not~loaded. \\
10812   You~can~t~use~the~key~'tikz'~here~because~TikZ~is~not~
10813   loaded.~If~you~go~on,~that~key~will~be~ignored.
10814 }
10815 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
10816 {
10817   TikZ~not~loaded. \\
10818   You~have~used~the~key~'tikz'~in~the~definition~of~a~
10819   customized~line~(with~'custom~line')~but~TikZ~is~not~loaded.~
10820   You~can~go~on~but~you~will~have~another~error~if~you~actually~
10821   use~that~custom~line.
10822 }
10823 \@@_msg_new:nn { tikz~in~borders~without~tikz }
10824 {
10825   TikZ~not~loaded. \\
10826   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
10827   command~' \token_to_str:N \Block ')~but~TikZ~is~not~loaded.~
10828   That~key~will~be~ignored.
10829 }
10830 \@@_msg_new:nn { color~in~custom~line~with~tikz }
10831 {
10832   Erroneous~use.\\
10833   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
10834   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
10835   The~key~'color'~will~be~discarded.
10836 }
10837 \@@_msg_new:nn { Wrong~last~row }
10838 {
10839   Wrong~number.\\
10840   You~have~used~'last~row= \int_use:N \l_@@_last_row_int '~but~your~
10841   \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
10842   If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
10843   last~row~but~you~should~correct~your~code.~You~can~avoid~this~
10844   problem~by~using~'last~row'~without~value~(more~compilations~
10845   might~be~necessary).
10846 }
10847 \@@_msg_new:nn { Yet~in~env }
10848 {
10849   Nested~environments.\\
10850   Environments~of~nicematrix~can't~be~nested.~However~you~
10851   can~insert,~for~example,~a~\{tabular\}~in~a~\{NiceTabular\}~
10852   or~a~\{NiceTabular\}~in~a~\{tabular\}.~You~can~also~compose~
10853   an~environment~of~nicematrix~in~a~box~of~LaTeX~and~insert~
10854   that~box~in~another~environment~of~nicematrix.\\
10855   This~error~is~fatal.
10856 }
10857 \@@_msg_new:nn { Outside~math~mode }
10858 {
10859   Outside~math~mode.\\
10860   The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
10861   (and~not~in~ \token_to_str:N \vcenter ).\\
10862   This~error~is~fatal.
10863 }
10864 \@@_msg_new:nn { One~letter~allowed }
10865 {
10866   Bad~name.\\
10867   The~value~of~key~' \l_keys_key_str '~must~be~of~length~1~and~
10868   you~have~used~' \l_keys_value_tl '~.\\
10869   It~will~be~ignored.
10870 }

```

```

10871 \@@_msg_new:nn { TabularNote~in~CodeAfter }
10872 {
10873   Environment~\{TabularNote\}~forbidden.\\
10874   You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
10875   but~*before*~the~ \token_to_str:N \CodeAfter . \\
10876   This~environment~\{TabularNote\}~will~be~ignored.
10877 }
10878 \@@_msg_new:nn { varwidth~not~loaded }
10879 {
10880   varwidth~not~loaded.\\
10881   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
10882   loaded.~You~should~load~'varwidth',~before~or~after~'nicematrix'. \\
10883   Your~column~will~behave~like~'p'.
10884 }
10885 \@@_msg_new:nn { varwidth~not~loaded~in~X }
10886 {
10887   varwidth~not~loaded.\\
10888   You~can't~use~the~key~'V'~in~your~column~'X'~
10889   because~'varwidth'~is~not~loaded.~You~should~load~'varwidth',~
10890   before~or~after~'nicematrix'.\\
10891   It~will~be~ignored. \\
10892 }
10893 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
10894 {
10895   Unknown~key.\\
10896   Your~key~' \l_keys_key_str '~is~unknown~for~a~rule.\\
10897   \c_@@_available_keys_str
10898 }
10899 {
10900   The~available~keys~are~(in~alphabetic~order):~
10901   color,~
10902   dotted,~
10903   multiplicity,~
10904   sep~color,~
10905   tikz,~and~total~width.
10906 }
10907
10908 \@@_msg_new:nnn { Unknown~key~for~Block }
10909 {
10910   Unknown~key. \\
10911   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10912   \token_to_str:N \Block . \\
10913   It~will~be~ignored. \\
10914   \c_@@_available_keys_str
10915 }
10916 {
10917   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10918   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~name,~
10919   opacity,~rounded~corners,~r,~respect~arraystretch,~rules/width,~t,~T,~tikz,~
10920   transparent~and~vlines.
10921 }
10922 \@@_msg_new:nnn { Unknown~key~for~Brace }
10923 {
10924   Unknown~key.\\
10925   The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
10926   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
10927   It~will~be~ignored. \\
10928   \c_@@_available_keys_str
10929 }
10930 {
10931   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
10932   right~shorten,~shorten~(which~fixes~both~left~shorten~and~

```

```

10933     right-shorten)~and-yshift.
10934 }
10935 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10936 {
10937     Unknown~key.\\
10938     The~key~' \l_keys_key_str '~is~unknown.\\
10939     It~will~be~ignored. \\
10940     \c_@@_available_keys_str
10941 }
10942 {
10943     The~available~keys~are~(in~alphabetic~order):~
10944     delimiters/color,~
10945     rules~(with~the~subkeys~'color'~and~'width'),~
10946     sub-matrix~(several~subkeys)~
10947     and~xdots~(several~subkeys).~
10948     The~latter~is~for~the~command~ \token_to_str:N \line .
10949 }
10950 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10951 {
10952     Unknown~key.\\
10953     The~key~' \l_keys_key_str '~is~unknown.\\
10954     It~will~be~ignored. \\
10955     \c_@@_available_keys_str
10956 }
10957 {
10958     The~available~keys~are~(in~alphabetic~order):~
10959     create-cell-nodes,~
10960     delimiters/color~and~
10961     sub-matrix~(several~subkeys).
10962 }
10963 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10964 {
10965     Unknown~key.\\
10966     The~key~' \l_keys_key_str '~is~unknown.\\
10967     That~key~will~be~ignored. \\
10968     \c_@@_available_keys_str
10969 }
10970 {
10971     The~available~keys~are~(in~alphabetic~order):~
10972     'delimiters/color',~
10973     'extra-height',~
10974     'hlines',~
10975     'hvlines',~
10976     'left-xshift',~
10977     'name',~
10978     'right-xshift',~
10979     'rules'~(with~the~subkeys~'color'~and~'width'),~
10980     'slim',~
10981     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10982     and~'right-xshift').\\
10983 }
10984 \@@_msg_new:nnn { Unknown~key~for~notes }
10985 {
10986     Unknown~key.\\
10987     The~key~' \l_keys_key_str '~is~unknown.\\
10988     That~key~will~be~ignored. \\
10989     \c_@@_available_keys_str
10990 }
10991 {
10992     The~available~keys~are~(in~alphabetic~order):~
10993     bottomrule,~
10994     code-after,~

```

```

10995     code-before(+),~
10996     detect-duplicates,~
10997     enumitem-keys,~
10998     enumitem-keys-para,~
10999     para,~
11000     label-in-list,~
11001     label-in-tabular~and~
11002     style.
11003 }

11004 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
11005 {
11006     Unknown~key.\\
11007     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11008     \token_to_str:N \RowStyle . \\
11009     That~key~will~be~ignored. \\
11010     \c_@@_available_keys_str
11011 }
11012 {
11013     The~available~keys~are~(in~alphabetic~order):~
11014     bold,~
11015     cell-space-top-limit(+),~
11016     cell-space-bottom-limit(+),~
11017     cell-space-limits(+),~
11018     color,~
11019     fill~(alias:~rowcolor),~
11020     nb-rows,~
11021     opacity~and~
11022     rounded-corners.
11023 }

11024 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
11025 {
11026     Unknown~key.\\
11027     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11028     \token_to_str:N \NiceMatrixOptions . \\
11029     That~key~will~be~ignored. \\
11030     \c_@@_available_keys_str
11031 }
11032 {
11033     The~available~keys~are~(in~alphabetic~order):~
11034     &~in~blocks,~
11035     allow-duplicate-names,~
11036     ampersand-in-blocks,~
11037     caption-above,~
11038     cell-space-bottom-limit(+),~
11039     cell-space-limits(+),~
11040     cell-space-top-limit(+),~
11041     code-for-first-col(+),~
11042     code-for-first-row(+),~
11043     code-for-last-col(+),~
11044     code-for-last-row(+),~
11045     corners,~
11046     custom-key,~
11047     create-extra-nodes,~
11048     create-medium-nodes,~
11049     create-large-nodes,~
11050     custom-line,~
11051     delimiters~(several~subkeys),~
11052     end-of-row,~
11053     first-col,~
11054     first-row,~
11055     hlines,~
11056     hvlines,~
11057     hvlines-except-borders,~

```



```

11058     last-col,~
11059     last-row,~
11060     left-margin,~
11061     light-syntax,~
11062     light-syntax-expanded,~
11063     matrix/columns-type,~
11064     no-cell-nodes,~
11065     notes~(several~subkeys),~
11066     nullify-dots,~
11067     pgf-node-code,~
11068     renew-dots,~
11069     renew-matrix,~
11070     respect-arraystretch,~
11071     rounded-corners,~
11072     right-margin,~
11073     rules~(with~the~subkeys~'color'~and~'width'),~
11074     small,~
11075     sub-matrix~(several~subkeys),~
11076     vlines,~
11077     xdots~(several~subkeys).
11078 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no `l` and `r`.

```

11079 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
11080 {
11081   Unknown~key.\\
11082   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11083   \{NiceArray\}. \\
11084   That~key~will~be~ignored. \\
11085   \c_@@_available_keys_str
11086 }
11087 {
11088   The~available~keys~are~(in~alphabetic~order):~
11089   &~in~blocks,~
11090   ampersand~in~blocks,~
11091   b,~
11092   baseline,~
11093   c,~
11094   cell-space-bottom-limit,~
11095   cell-space-limits,~
11096   cell-space-top-limit,~
11097   code~after,~
11098   code~for~first~col(+),~
11099   code~for~first~row(+),~
11100   code~for~last~col(+),~
11101   code~for~last~row(+),~
11102   columns~width,~
11103   corners,~
11104   create~blocks~in~col,~
11105   create~extra~nodes,~
11106   create~medium~nodes,~
11107   create~large~nodes,~
11108   extra~left~margin,~
11109   extra~right~margin,~
11110   first~col,~
11111   first~row,~
11112   hlines,~
11113   hvlines,~
11114   hvlines~except~borders,~
11115   last~col,~
11116   last~row,~
11117   left~margin,~
11118   light-syntax,~

```

```

11119 light-syntax-expanded,~
11120 name,~
11121 no-cell-nodes,~
11122 nullify-dots,~
11123 pgf-node-code,~
11124 renew-dots,~
11125 respect-arraystretch,~
11126 right-margin,~
11127 rounded-corners,~
11128 rules~(with~the~subkeys~'color'~and~'width'),~
11129 small,~
11130 t,~
11131 vlines,~
11132 xdots/color,~
11133 xdots/shorten-start(+),~
11134 xdots/shorten-end(+),~
11135 xdots/shorten(+)-and~
11136 xdots/line-style.
11137 }

```

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`).

```

11138 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
11139 {
11140   Unknown~key.\\
11141   The~key~' \l_keys_key_str '~is~unknown~for~the~
11142   \@@_full_name_env: . \\
11143   That~key~will~be~ignored. \\
11144   \c_@@_available_keys_str
11145 }
11146 {
11147   The~available~keys~are~(in~alphabetic~order):~
11148   &~in~blocks,~
11149   ampersand~in~blocks,~
11150   b,~
11151   baseline,~
11152   c,~
11153   cell-space-bottom-limit,~
11154   cell-space-limits,~
11155   cell-space-top-limit,~
11156   code-after,~
11157   code-for-first-col(+),~
11158   code-for-first-row(+),~
11159   code-for-last-col(+),~
11160   code-for-last-row(+),~
11161   columns-type,~
11162   columns-width,~
11163   corners,~
11164   create-blocks-in-col,~
11165   create-extra-nodes,~
11166   create-medium-nodes,~
11167   create-large-nodes,~
11168   extra-left-margin,~
11169   extra-right-margin,~
11170   first-col,~
11171   first-row,~
11172   hlines,~
11173   hvlines,~
11174   hvlines-except-borders,~
11175   l,~
11176   last-col,~
11177   last-row,~
11178   left-margin,~

```

```

11179 light-syntax,~
11180 light-syntax-expanded,~
11181 name,~
11182 no-cell-nodes,~
11183 nullify-dots,~
11184 pgf-node-code,~
11185 r,~
11186 renew-dots,~
11187 respect-arraystretch,~
11188 right-margin,~
11189 rounded-corners,~
11190 rules~(with~the~subkeys~'color'~and~'width'),~
11191 small,~
11192 t,~
11193 vlines,~
11194 xdots/color,~
11195 xdots/shorten-start(+),~
11196 xdots/shorten-end(+),~
11197 xdots/shorten(+)-and~
11198 xdots/line-style.
11199 }

11200 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
11201 {
11202   Unknown~key.\\
11203   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11204   \{NiceTabular\}. \\
11205   That~key~will~be~ignored. \\
11206   \c_@@_available_keys_str
11207 }
11208 {
11209   The~available~keys~are~(in~alphabetic~order):~
11210   &~in~blocks,~
11211   ampersand~in~blocks,~
11212   b,~
11213   baseline,~
11214   c,~
11215   caption,~
11216   cell-space-bottom-limit,~
11217   cell-space-limits,~
11218   cell-space-top-limit,~
11219   code-after,~
11220   code-for-first-col(+),~
11221   code-for-first-row(+),~
11222   code-for-last-col(+),~
11223   code-for-last-row(+),~
11224   columns-width,~
11225   corners,~
11226   custom-line,~
11227   create-blocks-in-col,~
11228   create-extra-nodes,~
11229   create-medium-nodes,~
11230   create-large-nodes,~
11231   extra-left-margin,~
11232   extra-right-margin,~
11233   first-col,~
11234   first-row,~
11235   hlines,~
11236   hvlines,~
11237   hvlines-except-borders,~
11238   label,~
11239   last-col,~
11240   last-row,~
11241   left-margin,~

```

```

11242 light-syntax,~
11243 light-syntax-expanded,~
11244 name,~
11245 no-cell-nodes,~
11246 notes~(several~subkeys),~
11247 nullify-dots,~
11248 pgf-node-code,~
11249 renew-dots,~
11250 respect-arraystretch,~
11251 right-margin,~
11252 rounded-corners,~
11253 rules~(with~the~subkeys~'color'~and~'width'),~
11254 short-caption,~
11255 t,~
11256 tabularnote,~
11257 vlines,~
11258 xdots/color,~
11259 xdots/shorten-start(+),~
11260 xdots/shorten-end(+),~
11261 xdots/shorten(+),~and~
11262 xdots/line-style.
11263 }
11264 \@@_msg_new:nnn { Duplicate-name }
11265 {
11266 Duplicate-name.\
11267 The-name-' \l_keys_value_tl 'is-already-used-and-you-shouldn't-use~
11268 the-same-environment-name-twice.~You-can-go-on,~but,~
11269 maybe,~you-will-have-incorrect-results-especially~
11270 if-you-use-'columns-width=auto'.~If-you-don't-want-to-see-this~
11271 message-again,~use-the-key~'allow-duplicate-names'~in~
11272 ' \token_to_str:N \NiceMatrixOptions '.\
11273 \bool_if:NF \g_@@_messages_for_Overleaf_bool
11274 { For-a-list-of-the-names-already-used,~type-H~<return>. }
11275 }
11276 {
11277 The-names~already-defined-in~this~document~are:~
11278 \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
11279 }
11280 \@@_msg_new:nn { caption-above-in-env }
11281 {
11282 The-key~'caption-above'~must-be-used-in~\token_to_str:N \NiceMatrixOptions.\
11283 That-key-will-be-ignored.
11284 }
11285 \@@_msg_new:nn { show-cell-names }
11286 {
11287 There-is-no-key~'show-cell-names'~in~nicematrix.\
11288 You-should-use-the-command~\token_to_str:N \ShowCellNames\
11289 in~the~\token_to_str:N \CodeBefore\ or~the~\token_to_str:N
11290 \CodeAfter. \
11291 That-key-will-be-ignored.
11292 }
11293 \@@_msg_new:nn { Option-auto-for-columns-width }
11294 {
11295 Erroneous~use.\
11296 You-can't-give-the-value~'auto'~to~the-key~'columns-width'~here.~
11297 That-key-will-be-ignored.
11298 }
11299 \@@_msg_new:nn { NiceTabularX~without~X }
11300 {
11301 NiceTabularX~without~X.\
11302 You-should-not-use~\{NiceTabularX\}~without~X~columns.\
11303 However,~you~can~go~on.

```

```

11304 }
11305 \@@_msg_new:nn { Preamble~forgotten }
11306 {
11307   Preamble~forgotten.\\
11308   You~have~probably~forgotten~the~preamble~of~your~
11309   \@@_full_name_env: . \\
11310   This~error~is~fatal.
11311 }
11312 \@@_msg_new:nn { Invalid~col~number }
11313 {
11314   Invalid~column~number.\\
11315   A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11316   specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.\\
11317   Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
11318 }
11319 \@@_msg_new:nn { Invalid~row~number }
11320 {
11321   Invalid~row~number.\\
11322   A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11323   specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.\\
11324   Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
11325 }
11326 \@@_define_com:NNN p ( )
11327 \@@_define_com:NNN b [ ]
11328 \@@_define_com:NNN v | |
11329 \@@_define_com:NNN V \l \l
11330 \@@_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>	38
9	The <code>\CodeBefore</code>	54
10	The environment <code>{NiceArrayWithDelims}</code>	58
11	Construction of the preamble of the array	64
12	The redefinition of <code>\multicolumn</code>	80
13	The environment <code>{NiceMatrix}</code> and its variants	98
	13.1 Definition of <code>{pNiceMatrix}</code>	98
	13.2 The key <code>renew-matrix</code>	99
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	99
15	After the construction of the array	100
16	We draw the dotted lines	107
17	The actual instructions for drawing the dotted lines with <code>TikZ</code>	124
18	User commands available in the new environments	130
19	The command <code>\line</code> accessible in <code>\CodeAfter</code>	137
20	The command <code>\RowStyle</code>	138
21	Colors of cells, rows and columns	141
22	The vertical and horizontal rules	153
23	The empty corners	171
24	The environment <code>{NiceMatrixBlock}</code>	173
25	The extra nodes	174
26	The blocks	179
27	Automatic arrays	205
28	The redefinition of the command <code>\dotfill</code>	207
29	The command <code>\diagbox</code>	207

30	The keyword <code>\CodeAfter</code>	209
31	The delimiters in the preamble	209
32	The command <code>\SubMatrix</code>	211
33	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	220
34	The commands <code>HBrace</code> et <code>VBrace</code>	223
35	The command <code>TikzEveryCell</code>	226
36	The command <code>\CreateBlocksInColumn</code>	227
37	The command <code>\ShowCellNames</code>	228
38	We process the options at package loading	229
39	About the package underscore	231
40	Compatibility with <code>threeparttable</code>	231
41	Error messages of the package	231