

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

February 1, 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 7.6 of `nicematrix`, at the date of 2026/02/01.

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

```

20 \RequirePackage { amsmath }

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

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

```

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

```

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

```

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

```

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

```

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

```

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

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

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

```

2 Collecting options

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

Example :

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

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

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

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

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

```

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

```

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

```

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

```

3 Technical definitions

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

```

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

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

```

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

```

83 \tl_new:N \l_@@_argspec_tl

```

```

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

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

170         \cs_set_eq:NN \rowcolor \@@_old_rowcolor:
171     }
172 }

```

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

```

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

```

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

```

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

```

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

```

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

```

Idem for `\CT@drs@`.

```

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

```

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

```

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

```

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

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

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

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

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

```
225 \cs_set:Npn \@@_cline:
```

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

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

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

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

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

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

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

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

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

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

¹See question 99041 on TeX StackExchange.

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```


4 Parameters

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
308 \dim_new:N \l_@@_col_width_dim
309 \dim_set:Nn \l_@@_col_width_dim { -1 cm }
```

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

```
310 \int_new:N \g_@@_row_total_int
311 \int_new:N \g_@@_col_total_int
```

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

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

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

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

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

```
314 \tl_new:N \l_@@_hpos_cell_tl
315 \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
```

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

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

Idem for the mono-row blocks.

```
317 \dim_new:N \g_@@_blocks_ht_dim
318 \dim_new:N \g_@@_blocks_dp_dim
```

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

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

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

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

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

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

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

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

```

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

```

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

```

327 \dim_new:N \l_@@_tabular_width_dim

```

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

```

328 \dim_new:N \l_@@_rule_width_dim

```

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

```

329 \tl_new:N \l_@@_rule_color_tl

```

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

```

330 \bool_new:N \g_@@_rotate_bool

```

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

```

331 \bool_new:N \g_@@_rotate_c_bool

```

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

```

332 \bool_new:N \l_@@_X_bool

```

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

```

333 \bool_new:N \l_@@_V_of_X_bool

```

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

```

334 \bool_new:N \g_@@_V_of_X_bool

```

```

335 \bool_new:N \g_@@_caption_finished_bool

```

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

```

336 \bool_new:N \l_@@_no_cell_nodes_bool

```

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

```

337 \tl_new:N \g_@@_aux_tl

```

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

```

338 \bool_new:N \g_@@_aux_found_bool

```

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

```

339 \seq_new:N \g_@@_size_seq

```

```

340 \tl_new:N \g_@@_left_delim_tl

```

```

341 \tl_new:N \g_@@_right_delim_tl

```

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

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

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

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

For `\multicolumn`.

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

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

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

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

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

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

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

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

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

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

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

```
352 {
```

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

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

```
355   \fi:
```

```
356 }
```

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

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

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

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

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

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

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

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

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

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

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

```
364 \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)`.

```

365 \cs_new:Npn \@@_full_name_env:
366 {
367   \str_if_eq:eeTF { \g_@@_com_or_env_str } { command }
368   { command \space \c_backslash_str \g_@@_name_env_str }
369   { environment \space \{ \g_@@_name_env_str \} }
370 }

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

```

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

```

372 \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.

```

373 \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.

```

374 \tl_new:N \g_@@_pre_code_before_tl
375 \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.

```

376 \tl_new:N \g_@@_pre_code_after_tl
377 \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.

```

378 \bool_new:N \l_@@_in_code_after_bool

```

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

```

379 \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.

```

380 \int_new:N \l_@@_old_iRow_int
381 \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).

```

382 \seq_new:N \l_@@_custom_line_commands_seq

```

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

```

383 \tl_new:N \l_@@_rules_color_tl

```

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

```
384 \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.

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

```
387 \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.

```
388 \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).

```
389 \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).

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

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

```
392 \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).

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

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

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

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

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

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

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

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

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

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

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

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

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

```
404 \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”.

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

```
406 \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.

```
407 \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!).

```
408 \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.

```
409 \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}`.

```
410 \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.

```
411 \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`).

```
412 \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`).

```
413 \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.

```
414 \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).

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

```
416 \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.

```
417 \int_new:N \g_@@_ddots_int
418 \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.

```
419 \dim_new:N \g_@@_delta_x_one_dim
420 \dim_new:N \g_@@_delta_y_one_dim
421 \dim_new:N \g_@@_delta_x_two_dim
422 \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).

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

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

The following counters will be used when drawing the rules.

```
431 \int_new:N \l_@@_start_int
432 \int_set_eq:NN \l_@@_start_int \c_one_int
433 \int_new:N \l_@@_end_int
434 \int_new:N \l_@@_local_start_int
435 \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.

```
436 \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).

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

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

```
438 \tl_new:N \l_@@_fill_tl
439 \tl_new:N \l_@@_opacity_tl
440 \tl_new:N \l_@@_draw_tl
441 \seq_new:N \l_@@_tikz_seq
442 \clist_new:N \l_@@_borders_clist
443 \dim_new:N \l_@@_rounded_corners_dim
```

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

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

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

```
444 \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`.

```
445 \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.

```
446 \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.

```
447 \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`).

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

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

```
452 \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).

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

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

```
454 \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).

```
455 \bool_new:N \l_@@_vlines_block_bool
456 \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.

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

458 \dim_new:N \l_@@_submatrix_extra_height_dim
459 \dim_new:N \l_@@_submatrix_left_xshift_dim
460 \dim_new:N \l_@@_submatrix_right_xshift_dim
461 \clist_new:N \l_@@_hlines_clist
462 \clist_new:N \l_@@_vlines_clist
463 \clist_new:N \l_@@_submatrix_hlines_clist
464 \clist_new:N \l_@@_submatrix_vlines_clist
```

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

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

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

```
466 \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`).

```
467 \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.

```
468 \int_new:N \l_@@_first_row_int
469 \int_set_eq:NN \l_@@_first_row_int \c_one_int
```

• First column

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

```
470 \int_new:N \l_@@_first_col_int
471 \int_set_eq:NN \l_@@_first_col_int \c_one_int
```

• Last row

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

```
472 \int_new:N \l_@@_last_row_int
473 \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

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

Idem for `\l_@@_last_col_without_value_bool`

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

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

- **Last column**

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

```
476 \int_new:N \l_@@_last_col_int
477 \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.

```
478 \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`).

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

Some utilities

```
480 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
481 {
```

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

```
482 \def \l_tmpa_tl { #1 }
483 \def \l_tmpb_tl { #2 }
484 }
```

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`.

```
485 \cs_new_protected:Npn \@@_expand_clist_hvlines:NN #1 #2
486 {
487   \clist_if_in:NnF #1 { all }
488   {
489     \clist_clear:N \l_tmpa_clist
490     \clist_map_inline:Nn #1
491     {
492       \tl_if_head_eq_meaning:nNTF { ##1 } -
493       {
```

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.

```
494       \int_if_zero:nF { #2 }
495       {
496         \clist_put_right:Ne \l_tmpa_clist
497         { \int_eval:n { #2 + (##1) + 1 } }
498       }
499     }
500   }
```

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

```

501         \tl_if_in:nnTF { ##1 } { - }
502         { \@@_cut_on_hyphen:w ##1 \q_stop }
503         {

```

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

```

504         \def \l_tmpa_tl { ##1 }
505         \def \l_tmpb_tl { ##1 }
506     }
507     \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
508     { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
509 }
510 }
511 \tl_set_eq:NN #1 \l_tmpa_clist
512 }
513 }

```

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.

```

514 \hook_gput_code:nnn { begindocument } { . }
515 {
516     \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
517     \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
518 }

```

5 The command `\tabularnote`

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

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

- The number of tabular notes present in the caption will be written on the aux file and available in `\g_@@_notes_caption_int`.⁴
- During the composition of the main tabular, the tabular notes will be numbered from `\g_@@_notes_caption_int+1` and the notes will be stored in `\g_@@_notes_seq`. Each component of `\g_@@_notes_seq` will be a kind of couple of the form : `{label}{text of the tabularnote}`. The first component is the optional argument (between square brackets) of the command `\tabularnote` (if the optional argument is not used, the value will be the special marker expressed by `\NoValue`).

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

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

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

```
519 \newcounter { tabularnote }
```

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

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

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

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

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

```
525 \seq_new:N \l_@@_notes_labels_seq
526 \newcounter { nicematrix_draft }
527 \cs_new_protected:Npn \@@_notes_format:n #1
528 {
529   \setcounter { nicematrix_draft } { #1 }
530   \@@_notes_style:n { nicematrix_draft }
531 }
```

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

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

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

```
533 \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`.

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

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

```
535 \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`).

```
536 \hook_gput_code:nnn { begindocument } { . }
537 {
538   \IfPackageLoadedTF { enumitem }
539   {
```

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.

```

540     \newlist { tabularnotes } { enumerate } { 1 }
541     \setlist [ tabularnotes ]
542     {
543         topsep = \c_zero_dim ,
544         noitemsep ,
545         leftmargin = * ,
546         align = left ,
547         labelsep = \c_zero_dim ,
548         label =
549             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
550     }
551     \newlist { tabularnotes* } { enumerate* } { 1 }
552     \setlist [ tabularnotes* ]
553     {
554         afterlabel = \nobreak ,
555         itemjoin = \quad ,
556         label =
557             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
558     }

```

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`.

```

559     \NewDocumentCommand \tabularnote { o m }
560     {
561         \bool_lazy_or:nnT { \cs_if_exist_p:N \@capttype } { \l_@@_in_env_bool }
562         {
563             \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } { \l_@@_in_env_bool }
564             { \@@_error:n { tabularnote~forbidden } }
565             {
566                 \bool_if:NTF \l_@@_in_caption_bool
567                 \@@_tabularnote_caption:nn
568                 \@@_tabularnote:nn
569                 { #1 } { #2 }
570             }
571         }
572     }
573 }
574 {
575     \NewDocumentCommand \tabularnote { o m }
576     { \@@_err_enumitem_not_loaded: }
577 }
578 }
579 \cs_new_protected:Npn \@@_err_enumitem_not_loaded:
580 {
581     \@@_error_or_warning:n { enumitem~not~loaded }
582     \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
583 }
584 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
585 { \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`.

```

586 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
587 {

```

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).

```

588     \int_zero:N \l_tmpa_int
589     \bool_if:NT \l_@@_notes_detect_duplicates_bool
590     {

```

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`.

```

591     \int_zero:N \l_tmpb_int
592     \seq_map_indexed_inline:Nn \g_@@_notes_seq
593     {
594         \@@_test_first_novaluennn ##2 { \int_incr:N \l_tmpb_int }
595         \tl_if_eq:nnT { { #1 } { #2 } } { { ##2 } }
596         {
597             \tl_if_novalue:nTF { #1 }
598             { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
599             { \int_set:Nn \l_tmpa_int { ##1 } }
600             \seq_map_break:
601         }
602     }
603     \int_if_zero:nF { \l_tmpa_int }
604     { \int_add:Nn \l_tmpa_int { \g_@@_notes_caption_int } }
605 }
606 \int_if_zero:nT { \l_tmpa_int }
607 {
608     \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
609     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
610 }
611 \seq_put_right:Ne \l_@@_notes_labels_seq
612 {
613     \tl_if_novalue:nTF { #1 }
614     {
615         \@@_notes_format:n
616         {
617             \int_eval:n
618             {
619                 \int_if_zero:nTF { \l_tmpa_int }
620                 { \c@tabularnote }
621                 { \l_tmpa_int }
622             }
623         }
624     }
625     { #1 }
626 }
627 \peek_meaning:NF \tabularnote
628 {

```

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`.

```

629     \hbox_set:Nn \l_tmpa_box
630     {

```

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

```

631         \@@_notes_label_in_tabular:n
632         { \seq_use:Nn \l_@@_notes_labels_seq { , } }
633     }

```

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

```

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

```

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

```

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

```

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

```

646         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
647     }
648     { \box_use:N \l_tmpa_box }
649 }
650 }

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

663         \bool_gset_true:N \g_@@_caption_finished_bool
664         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
665         \int_gzero:N \c@tabularnote
666     }

```



```

667         { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
668     }

```

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

```

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

```

```

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

```

6 Command for creation of rectangle nodes

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

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

```

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

```

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

```

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

```

```

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

```

7 The options

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

```

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

```

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

```

734 \bool_new:N \l_@@_caption_above_bool

```

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

```

735 \bool_new:N \l_@@_standard_cline_bool

```

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

```

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

```

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

```

738 \bool_new:N \l_@@_xdots_h_labels_bool

```

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

```

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

```

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

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

```

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

```

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

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

759 \bool_new:N \l_@@_amp_in_blocks_bool

```

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

```

760 \bool_new:N \l_@@_exterior_arraycolsep_bool

```

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

```

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

```

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

```

763 \clist_new:N \l_@@_corners_clist

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

780 \bool_new:N \l_@@_delimiters_max_width_bool

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

```

```

834     { \@@_error:n { bad-option-for-line-style } }
835   } ,
836   line-style .value_required:n = true ,
837   color .tl_set:N = \l_@@_xdots_color_tl ,
838   color .value_required:n = true ,
839   radius .code:n =
840     \hook_gput_code:nnn { begindocument } { . }
841     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
842   radius .value_required:n = true ,
843   inter .code:n =
844     \hook_gput_code:nnn { begindocument } { . }
845     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
846   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 \wedge , $_$ and \cdot . 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 `\wedge{...}`.

```

847   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
848   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
849   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).

```

850   draw-first .code:n = \prg_do_nothing: ,
851   unknown .code:n =
852     \@@_unknown_key:nn { nicematrix / xdots } { Unknown-key-for-xdots }
853 }

```

```

854 \keys_define:nn { nicematrix / rules }
855 {
856   color .tl_set:N = \l_@@_rules_color_tl ,
857   color .value_required:n = true ,
858   width .dim_set:N = \arrayrulewidth ,
859   width .value_required:n = true ,
860   unknown .code:n = \@@_error:n { Unknown-key-for-rules }
861 }

```

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

```

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

```

```

884 cell-space-top-limit .value_required:n = true ,
885 cell-space-top-limit+ .code:n =
886   \dim_add:Nn \l_@@_cell_space_top_limit_dim { #1 } ,
887 cell-space-top-limit+ .value_required:n = true ,
888 cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
889 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
890 cell-space-bottom-limit .value_required:n = true ,
891 cell-space-bottom-limit+ .code:n =
892   \dim_add:Nn \l_@@_cell_space_bottom_limit_dim { #1 } ,
893 cell-space-bottom-limit+ .value_required:n = true ,
894 cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
895 cell-space-limits .meta:n =
896   {
897     cell-space-top-limit = #1 ,
898     cell-space-bottom-limit = #1 ,
899   } ,
900 cell-space-limits .value_required:n = true ,
901 cell-space-limits+ .meta:n =
902   {
903     cell-space-top-limit += #1 ,
904     cell-space-bottom-limit += #1 ,
905   } ,
906 cell-space-limits+ .value_required:n = true ,
907 cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
908 xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
909 light-syntax .code:n =
910   \bool_set_true:N \l_@@_light_syntax_bool
911   \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
912 light-syntax .value_forbidden:n = true ,
913 light-syntax-expanded .code:n =
914   \bool_set_true:N \l_@@_light_syntax_bool
915   \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
916 light-syntax-expanded .value_forbidden:n = true ,
917 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
918 end-of-row .value_required:n = true ,
919
920 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
921 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
922 last-row .int_set:N = \l_@@_last_row_int ,
923 last-row .default:n = -1 ,
924
925 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
926 code-for-first-col .value_required:n = true ,
927 code-for-first-col+ .code:n =
928   { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
929 code-for-first-col+ .value_required:n = true ,
930 code-for-first-col~+ .meta:n = { code-for-first-col+ = #1 } ,
931
932 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
933 code-for-last-col .value_required:n = true ,
934 code-for-last-col+ .code:n =
935   { \tl_put_right:Nn \l_@@_code_for_last_col_tl { #1 } } ,
936 code-for-last-col+ .value_required:n = true ,
937 code-for-last-col~+ .meta:n = { code-for-last-col+ = #1 } ,
938
939 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
940 code-for-first-row .value_required:n = true ,
941 code-for-first-row+ .code:n =
942   { \tl_put_right:Nn \l_@@_code_for_first_row_tl { #1 } } ,
943 code-for-first-row+ .value_required:n = true ,
944 code-for-first-row~+ .meta:n = { code-for-first-row+ = #1 } ,
945
946 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,

```

```

947 code-for-last-row .value_required:n = true ,
948 code-for-last-row+ .code:n =
949   { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
950 code-for-last-row+ .value_required:n = true ,
951 code-for-last-row+ .meta:n = { code-for-last-row+ = #1 } ,
952
953 hlines .clist_set:N = \l_@@_hlines_clist ,
954 vlines .clist_set:N = \l_@@_vlines_clist ,
955 hlines .default:n = all ,
956 vlines .default:n = all ,
957 vlines-in-sub-matrix .code:n =
958   {
959     \tl_if_single_token:nTF { #1 }
960     {
961       \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
962       { \@@_error:nn { Forbidden-letter } { #1 } }

```

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

```

963     { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
964   }
965   { \@@_error:n { One-letter-allowed } }
966 } ,
967 vlines-in-sub-matrix .value_required:n = true ,
968 hvlines .code:n =
969   {
970     \bool_set_true:N \l_@@_hvlines_bool
971     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
972     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
973   } ,
974 hvlines .value_forbidden:n = true ,
975 hvlines-except-borders .code:n =
976   {
977     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
978     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
979     \bool_set_true:N \l_@@_hvlines_bool
980     \bool_set_true:N \l_@@_except_borders_bool
981   } ,
982 hvlines-except-borders .value_forbidden:n = true ,
983 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.

```

984 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
985 renew-dots .value_forbidden:n = true ,
986 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
987 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
988 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
989 create-extra-nodes .meta:n =
990   { create-medium-nodes , create-large-nodes } ,
991 left-margin .dim_set:N = \l_@@_left_margin_dim ,
992 left-margin .default:n = \arraycolsep ,
993 right-margin .dim_set:N = \l_@@_right_margin_dim ,
994 right-margin .default:n = \arraycolsep ,
995 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
996 margin .default:n = \arraycolsep ,
997 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
998 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
999 extra-margin .meta:n =
1000   { extra-left-margin = #1 , extra-right-margin = #1 } ,
1001 extra-margin .value_required:n = true ,
1002 respect-arraystretch .code:n =
1003   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
1004 respect-arraystretch .value_forbidden:n = true ,

```



```

1005     pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
1006     pgf-node-code .value_required:n = true
1007 }

```

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

```

1008 \keys_define:nn { nicematrix / environments }
1009 {
1010     create-blocks-in-col .code:n = \@@_create_blocks_in_col:n { #1 } ,
1011     create-blocks-in-col .value_required:n = true ,
1012     corners .clist_set:N = \l_@@_corners_clist ,
1013     corners .default:n = { NW , SW , NE , SE } ,
1014     code-before .code:n =
1015     {
1016         \tl_if_empty:nF { #1 }
1017         {
1018             \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1019             \bool_set_true:N \l_@@_code_before_bool
1020         }
1021     } ,
1022     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}`.

```

1023     c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
1024     t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
1025     b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
1026     baseline .tl_set:N = \l_@@_baseline_tl ,
1027     baseline .value_required:n = true ,
1028     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)`.

```

1029     \str_if_eq:eeTF { #1 } { auto }
1030     { \bool_set_true:N \l_@@_auto_columns_width_bool }
1031     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
1032     columns-width .value_required:n = true ,
1033     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.

```

1034     \legacy_if:nF { measuring@ }
1035     {
1036         \str_set:Nn \l_@@_name_str { #1 }
1037         \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
1038         { \@@_err_duplicate_names:n { #1 } }
1039         { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
1040     } ,
1041     name .value_required:n = true ,
1042     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
1043     code-after .value_required:n = true ,
1044 }

1045 \cs_set:Npn \@@_err_duplicate_names:n #1
1046 { \@@_error:nn { Duplicate-name } { #1 } }

1047 \keys_define:nn { nicematrix / notes }
1048 {
1049     para .bool_set:N = \l_@@_notes_para_bool ,
1050     para .default:n = true ,
1051     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
1052     code-before .value_required:n = true ,
1053     code-before+ .code:n =
1054     \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,

```

```

1055 code-before+ .value_required:n = true ,
1056 code-before~+ .code:n =
1057   \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1058 code-before~+ .value_required:n = true ,
1059 code-after .tl_set:N = \l_@@_notes_code_after_tl ,
1060 code-after .value_required:n = true ,
1061 bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
1062 bottomrule .default:n = true ,
1063 style .cs_set:Np = \@@_notes_style:n #1 ,
1064 style .value_required:n = true ,
1065 label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
1066 label-in-tabular .value_required:n = true ,
1067 label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1068 label-in-list .value_required:n = true ,
1069 enumitem-keys .code:n =
1070 {
1071   \hook_gput_code:nnn { begindocument } { . }
1072   {
1073     \IfPackageLoadedT { enumitem }
1074     { \setlist* [ tabularnotes ] { #1 } }
1075   }
1076 } ,
1077 enumitem-keys .value_required:n = true ,
1078 enumitem-keys-para .code:n =
1079 {
1080   \hook_gput_code:nnn { begindocument } { . }
1081   {
1082     \IfPackageLoadedT { enumitem }
1083     { \setlist* [ tabularnotes* ] { #1 } }
1084   }
1085 } ,
1086 enumitem-keys-para .value_required:n = true ,
1087 detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1088 detect-duplicates .default:n = true ,
1089 unknown .code:n =
1090   \@@_unknown_key:nn { nicematrix / notes } { Unknown~key~for~notes }
1091 }
1092 \keys_define:nn { nicematrix / delimiters }
1093 {
1094   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1095   max-width .default:n = true ,
1096   color .tl_set:N = \l_@@_delimiters_color_tl ,
1097   color .value_required:n = true ,
1098 }

```

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

```

1099 \keys_define:nn { nicematrix }
1100 {
1101   NiceMatrixOptions .inherit:n =
1102     { nicematrix / Global } ,
1103   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1104   NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1105   NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1106   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1107   SubMatrix / rules .inherit:n = nicematrix / rules ,
1108   CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1109   CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1110   CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1111   NiceMatrix .inherit:n =
1112     {
1113       nicematrix / Global ,

```

```

1114     nicematrix / environments ,
1115   } ,
1116   NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1117   NiceMatrix / rules .inherit:n = nicematrix / rules ,
1118   NiceTabular .inherit:n =
1119   {
1120     nicematrix / Global ,
1121     nicematrix / environments
1122   } ,
1123   NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1124   NiceTabular / rules .inherit:n = nicematrix / rules ,
1125   NiceTabular / notes .inherit:n = nicematrix / notes ,
1126   NiceArray .inherit:n =
1127   {
1128     nicematrix / Global ,
1129     nicematrix / environments ,
1130   } ,
1131   NiceArray / xdots .inherit:n = nicematrix / xdots ,
1132   NiceArray / rules .inherit:n = nicematrix / rules ,
1133   pNiceArray .inherit:n =
1134   {
1135     nicematrix / Global ,
1136     nicematrix / environments ,
1137   } ,
1138   pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1139   pNiceArray / rules .inherit:n = nicematrix / rules ,
1140 }

```

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

```

1141 \keys_define:nn { nicematrix / NiceMatrixOptions }
1142 {
1143   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1144   delimiters / color .value_required:n = true ,
1145   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1146   delimiters / max-width .default:n = true ,
1147   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1148   delimiters .value_required:n = true ,
1149   width .dim_set:N = \l_@@_width_dim ,
1150   width .value_required:n = true ,
1151   last-col .code:n =
1152     \tl_if_empty:nF { #1 }
1153     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1154     \int_zero:N \l_@@_last_col_int ,
1155   small .bool_set:N = \l_@@_small_bool ,
1156   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.

```

1157   renew-matrix .code:n = \@@_renew_matrix: ,
1158   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}`.

```

1159   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.

```

1160   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)`.

```

1161     \str_if_eq:eeTF { #1 } { auto }
1162     { \@@_error:n { Option~auto~for~columns~width } }
1163     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1164     allow-duplicate-names .code:n =
1165     \cs_set:Nn \@@_err_duplicate_names:n { } ,
1166     allow-duplicate-names .value_forbidden:n = true ,
1167     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1168     notes .value_required:n = true ,
1169     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1170     sub-matrix .value_required:n = true ,
1171     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1172     matrix / columns-type .value_required:n = true ,
1173     caption-above .bool_set:N = \l_@@_caption_above_bool ,
1174     caption-above .default:n = true ,
1175     unknown .code:n =
1176     \@@_unknown_key:nn
1177     { nicematrix / Global , nicematrix / NiceMatrixOptions }
1178     { Unknown-key-for-NiceMatrixOptions }
1179 }

```

`\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.

```

1180 \NewDocumentCommand \NiceMatrixOptions { m }
1181 { \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.

```

1182 \keys_define:nn { nicematrix / NiceMatrix }
1183 {
1184     last-col .code:n = \tl_if_empty:nTF { #1 }
1185     {
1186         \bool_set_true:N \l_@@_last_col_without_value_bool
1187         \int_set:Nn \l_@@_last_col_int { -1 }
1188     }
1189     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1190     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1191     columns-type .value_required:n = true ,
1192     l .meta:n = { columns-type = l } ,
1193     r .meta:n = { columns-type = r } ,
1194     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1195     delimiters / color .value_required:n = true ,
1196     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1197     delimiters / max-width .default:n = true ,
1198     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1199     delimiters .value_required:n = true ,
1200     small .bool_set:N = \l_@@_small_bool ,
1201     small .value_forbidden:n = true ,
1202     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1203 }

```

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

```

1204 \keys_define:nn { nicematrix / NiceArray }
1205 {

```

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.

```

1206     small .bool_set:N = \l_@@_small_bool ,
1207     small .value_forbidden:n = true ,
1208     last-col .code:n = \tl_if_empty:nF { #1 }
1209         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1210         \int_zero:N \l_@@_last_col_int ,
1211     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1212     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1213     unknown .code:n =
1214         \@@_unknown_key:nn
1215         { nicematrix / NiceArray , nicematrix / Global , nicematrix / environments }
1216         { Unknown-key-for-NiceArray }
1217 }

1218 \keys_define:nn { nicematrix / pNiceArray }
1219 {
1220     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1221     last-col .code:n = \tl_if_empty:nF { #1 }
1222         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1223         \int_zero:N \l_@@_last_col_int ,
1224     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1225     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1226     delimiters / color .value_required:n = true ,
1227     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1228     delimiters / max-width .default:n = true ,
1229     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1230     delimiters .value_required:n = true ,
1231     small .bool_set:N = \l_@@_small_bool ,
1232     small .value_forbidden:n = true ,
1233     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1234     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1235     unknown .code:n =
1236         \@@_unknown_key:nn
1237         { nicematrix / pNiceArray , nicematrix / Global , nicematrix / environments }
1238         { Unknown-key-for-NiceMatrix }
1239 }

```

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

```

1240 \keys_define:nn { nicematrix / NiceTabular }
1241 {

```

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.

```

1242     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1243         \bool_set_true:N \l_@@_width_used_bool ,
1244     width .value_required:n = true ,
1245     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1246     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1247     tabularnote .value_required:n = true ,
1248     caption .tl_set:N = \l_@@_caption_tl ,
1249     caption .value_required:n = true ,
1250     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1251     short-caption .value_required:n = true ,
1252     label .tl_set:N = \l_@@_label_tl ,
1253     label .value_required:n = true ,
1254     last-col .code:n = \tl_if_empty:nF { #1 }
1255         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1256         \int_zero:N \l_@@_last_col_int ,
1257     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1258     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1259     unknown .code:n =

```

```

1260 \@@_unknown_key:nn
1261 { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
1262 { Unknown~key~for~NiceTabular }
1263 }

```

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
```

```

1264 \keys_define:nn { nicematrix / CodeAfter }
1265 {
1266   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1267   delimiters / color .value_required:n = true ,
1268   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1269   rules .value_required:n = true ,
1270   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1271   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1272   sub-matrix .value_required:n = true ,
1273   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1274 }

```

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}`).

```

1275 \cs_new_protected:Npn \@@_cell_begin:
1276 {

```

`\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.

```
1277 \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).

```
1278 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1279 \cs_set_eq:NN \Hline \@@_Hline_in_cell:
```

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

```
1280 \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.

```

1281 \if_int_compare:w \c@jCol = \c_one_int
1282 \if_int_compare:w \l_@@_first_col_int = \c_one_int
1283 \@@_begin_of_row:
1284 \fi:
1285 \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:`.

```
1286 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1287 \@@_tuning_not_tabular_begin:
1288 \@@_tuning_first_row:
1289 \@@_tuning_last_row:
1290 \g_@@_row_style_tl
1291 }
```

The following command will be nullified unless there is a first row. Here is a version with the standard syntax of L3.

```
\cs_new_protected:Npn \@@_tuning_first_row:
{
  \int_if_zero:nTF { \c@iRow }
  {
    \int_if_zero:nF { \c@jCol }
    {
      \l_@@_code_for_first_row_tl
      \xglobal \colorlet { nicematrix-first-row } { . }
    }
  }
  { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
}
```

We will use a version a little more efficient.

```
1292 \cs_new_protected:Npn \@@_tuning_first_row:
1293 {
1294   \if_int_compare:w \c@iRow = \c_zero_int
1295     \if_int_compare:w \c@jCol > \c_zero_int
1296       \l_@@_code_for_first_row_tl
1297       \xglobal \colorlet { nicematrix-first-row } { . }
1298     \fi:
1299   \else:
1300     \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing:
1301   \fi:
1302 }
```

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.

```
1303 \cs_new_protected:Npn \@@_tuning_last_row:
1304 {
1305   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1306     \l_@@_code_for_last_row_tl
1307     \xglobal \colorlet { nicematrix-last-row } { . }
1308   \fi:
1309 }
```

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

```
1310 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```

1311 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1312 {
1313     \m@th
1314     $ % $

```

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

```

1315     \@@_tuning_key_small:
1316 }
1317 \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.

```

1318 \cs_new_protected:Npn \@@_begin_of_row:
1319 {
1320     \int_gincr:N \c@iRow
1321     \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1322     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1323     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1324     \pgfpicture
1325     \pgfrememberpicturepositiononpagetrue
1326     \pgfcoordinate
1327     { \@@_env: - row - \int_use:N \c@iRow - base }
1328     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1329     \str_if_empty:NF \l_@@_name_str
1330     {
1331         \pgfnodealias
1332         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1333         { \@@_env: - row - \int_use:N \c@iRow - base }
1334     }
1335     \endpgfpicture
1336 }

```

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.

```

1337 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1338 {
1339   \if_int_compare:w \c@iRow = \c_zero_int
1340     \if_dim:w \box_dp:N \l_@@_cell_box > \g_@@_dp_row_zero_dim
1341       \global \g_@@_dp_row_zero_dim = \box_dp:N \l_@@_cell_box
1342     \fi:
1343     \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_zero_dim
1344       \global \g_@@_ht_row_zero_dim = \box_ht:N \l_@@_cell_box
1345     \fi:
1346   \else:
1347     \if_int_compare:w \c@iRow = \c_one_int
1348       \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_one_dim
1349         \global \g_@@_ht_row_one_dim = \box_ht:N \l_@@_cell_box
1350       \fi:
1351     \fi:
1352   \fi:
1353 }

1354 \cs_new_protected:Npn \@@_rotate_cell_box:
1355 {
1356   \box_rotate:Nn \l_@@_cell_box { 90 }
1357   \bool_if:NTF \g_@@_rotate_c_bool
1358   {
1359     \hbox_set:Nn \l_@@_cell_box
1360     {
1361       \m@th
1362       $ % $
1363       \vcenter { \box_use:N \l_@@_cell_box }
1364       $ % $
1365     }
1366   }
1367   {
1368     \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
1369     {
1370       \vbox_set_top:Nn \l_@@_cell_box
1371       {
1372         \vbox_to_zero:n { }
1373         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1374         \box_use:N \l_@@_cell_box
1375       }
1376     }
1377   }
1378   \bool_gset_false:N \g_@@_rotate_bool
1379   \bool_gset_false:N \g_@@_rotate_c_bool
1380 }

```

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.

```

1381 \cs_set_protected:Npn \@@_adjust_size_box:
1382 {
1383   \if_dim:w \g_@@_blocks_wd_dim > \c_zero_dim
1384     \if_dim:w \g_@@_blocks_wd_dim > \box_wd:N \l_@@_cell_box
1385       \box_wd:N \l_@@_cell_box = \g_@@_blocks_wd_dim
1386     \fi:
1387     \global \g_@@_blocks_wd_dim = \c_zero_dim
1388   \fi:
1389   \if_dim:w \g_@@_blocks_dp_dim > \c_zero_dim
1390     \if_dim:w \g_@@_blocks_dp_dim > \box_dp:N \l_@@_cell_box
1391       \box_dp:N \l_@@_cell_box = \g_@@_blocks_dp_dim
1392     \fi
1393     \global \g_@@_blocks_dp_dim = \c_zero_dim
1394   \fi:
1395   \if_dim:w \g_@@_blocks_ht_dim > \c_zero_dim
1396     \if_dim:w \g_@@_blocks_ht_dim > \box_ht:N \l_@@_cell_box
1397       \box_ht:N \l_@@_cell_box = \g_@@_blocks_ht_dim
1398     \fi:
1399     \global \g_@@_blocks_ht_dim = \c_zero_dim
1400   \fi:
1401 }
1402 \cs_new_protected:Npn \@@_cell_end:
1403 {

```

The following command is nullified in the tabulars.

```

1404   \@@_tuning_not_tabular_end:
1405   \hbox_set_end:
1406   \@@_cell_end_i:
1407 }

```

```

\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.

```

1408 \cs_new_protected:Npn \@@_cell_end_i:
1409 {

```

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.

```

1410 \g_@@_cell_after_hook_tl
1411 \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
1412 \@@_adjust_size_box:
1413 \box_set_ht:Nn \l_@@_cell_box
1414 { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1415 \box_set_dp:Nn \l_@@_cell_box
1416 { \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”).

```

1417 \@@_update_max_cell_width:

```

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

```

1418 \@@_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.

```

1419 \bool_if:NTF \g_@@_empty_cell_bool
1420 { \box_use_drop:N \l_@@_cell_box }
1421 {
1422 \bool_if:NTF \g_@@_not_empty_cell_bool
1423 { \@@_print_node_cell: }
1424 {
1425 \if_dim:w \box_wd:N \l_@@_cell_box > \c_zero_dim
1426 \@@_print_node_cell:
1427 \else:
1428 \box_use_drop:N \l_@@_cell_box
1429 \fi:
1430 }
1431 }
1432 \if_int_compare:w \c@jCol > \g_@@_col_total_int
1433 \global \g_@@_col_total_int = \c@jCol
1434 \fi:
1435 \global \let \g_@@_empty_cell_bool \c_false_bool
1436 \global \let \g_@@_not_empty_cell_bool \c_false_bool
1437 }

```

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:

```
1438 \cs_new_protected:Npn \@@_update_max_cell_width:
1439 {
1440   \if_dim:w \box_wd:N \l_@@_cell_box > \g_@@_max_cell_width_dim
1441     \global \g_@@_max_cell_width_dim = \box_wd:N \l_@@_cell_box
1442   \fi:
1443 }
```

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`).

```
1444 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1445 {
1446   \@@_math_toggle:
1447   \hbox_set_end:
1448   \bool_if:NF \g_@@_rotate_bool
1449   {
1450     \hbox_set:Nn \l_@@_cell_box
1451     {
1452       \makebox [ \l_@@_col_width_dim ] [ s ]
1453       { \hbox_unpack_drop:N \l_@@_cell_box }
1454     }
1455   }
1456   \@@_cell_end_i:
1457 }
```

```
1458 \pgfset
1459 {
1460   nicematrix / cell-node /.style =
1461   {
1462     inner~sep = \c_zero_dim ,
1463     minimum~width = \c_zero_dim
1464   }
1465 }
```

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.

```
1466 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1467 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1468 {
1469   \use:c
1470   {
1471     __siunitx_table_align_
1472     \bool_if:NTF \l__siunitx_table_text_bool
1473       { \l__siunitx_table_align_text_tl }
1474       { \l__siunitx_table_align_number_tl }
1475     :n
1476   }
1477   { #1 }
1478 }
```

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!

```

1479 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1480 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1481 {
1482   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1483   \hbox:n
1484   {
1485     \pgfsys@markposition
1486     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1487   }
1488   #1
1489   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1490   \hbox:n
1491   {
1492     \pgfsys@markposition
1493     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1494   }
1495 }

1496 \cs_new_protected:Npn \@@_print_node_cell:
1497 {
1498   \socket_use:nn { nicematrix / siunitx-wrap }
1499   { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1500 }

```

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

```

1501 \cs_new_protected:Npn \@@_node_cell:
1502 {
1503   \pgfpicture
1504   \pgfsetbaseline \c_zero_dim
1505   \pgfrememberpicturepositiononpagetrue
1506   \pgfset { nicematrix / cell-node }
1507   \pgfnode
1508   { rectangle }
1509   { base }
1510   {

```

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).

```

1511   \sys_if_engine_xetex:T { \set@color }
1512   \box_use:N \l_@@_cell_box
1513 }
1514 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1515 { \l_@@_pgf_node_code_tl }
1516 \str_if_empty:NF \l_@@_name_str
1517 {
1518   \pgfnodealias
1519   { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1520   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1521 }
1522 \endpgfpicture
1523 }

```

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).

```
1524 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1525 {
1526   \bool_if:nTF { #1 } { \tl_gput_left:ce } { \tl_gput_right:ce }
1527   { g_@@_ #2 _ lines _ tl }
1528   {
1529     \use:c { @@ _ draw _ #2 : nnn }
1530     { \int_use:N \c@iRow }
1531     { \int_use:N \c@jCol }
1532     { \exp_not:n { #3 } }
1533   }
1534 }

1535 \cs_new_protected:Npn \@@_array:n
1536 {
1537   \dim_set:Nn \col@sep
1538   { \bool_if:nTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1539   \dim_compare:nNnTF { \l_@@_tabular_width_dim } = { \c_zero_dim }
1540   { \def \@halignto { } }
1541   { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }
```

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

```
1542   \@tabarray

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

1543   [ \str_if_eq:eeTF { \l_@@_baseline_tl } { c } { c } { t } ]
1544   }
1545   \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`.

```
1546 \bool_if:nTF \c_@@_revtex_bool
1547 { \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`.

```
1548 { \cs_new_eq:cN { @@_old_ar@ialign: } \ar@ialign }
```

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

```
1549 \cs_new_protected:Npn \@@_create_row_node:
1550 {
1551   \int_compare:nNnT { \c@iRow } > { \g_@@_last_row_node_int }
1552   {
1553     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1554     \@@_create_row_node_i:
1555   }
1556 }
```

```

1557 \cs_new_protected:Npn \@@_create_row_node_i:
1558 {

```

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

```

1559     \hbox
1560     {
1561         \bool_if:NT \l_@@_code_before_bool
1562         {
1563             \vtop
1564             {
1565                 \skip_vertical:N 0.5\arrayrulewidth
1566                 \pgfsys@markposition
1567                 { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1568                 \skip_vertical:N -0.5\arrayrulewidth
1569             }
1570         }
1571         \pgfpicture
1572         \pgfrememberpicturepositiononpagetrue
1573         \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1574         { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1575         \str_if_empty:NF \l_@@_name_str
1576         {
1577             \pgfnodealias
1578             { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1579             { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1580         }
1581         \endpgfpicture
1582     }
1583 }

```

```

1584 \cs_new_protected:Npn \@@_in_everycr:
1585 {
1586     \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1587     \tbl_update_cell_data_for_next_row:
1588     \int_gzero:N \c@jCol
1589     \bool_gset_false:N \g_@@_after_col_zero_bool
1590     \bool_if:NF \g_@@_row_of_col_done_bool
1591     {
1592         \@@_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).

```

1593         \clist_if_empty:NF \l_@@_hlines_clist
1594         {
1595             \str_if_eq:eeF { \l_@@_hlines_clist } { all }
1596             {
1597                 \clist_if_in:NcT
1598                 \l_@@_hlines_clist
1599                 { \int_eval:n { \c@iRow + 1 } }
1600             }
1601         }

```

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.

```

1602             \int_compare:nNnT { \c@iRow } > { -1 }
1603             {
1604                 \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
1605                 { \hrule height \arrayrulewidth width \c_zero_dim }
1606             }
1607         }
1608     }
1609 }
1610 }

```

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

```

1611 \cs_set_protected:Npn \@@_renew_dots:
1612 {
1613   \cs_set_eq:NN \ldots \@@_Ldots:
1614   \cs_set_eq:NN \cdots \@@_Cdots:
1615   \cs_set_eq:NN \vdots \@@_Vdots:
1616   \cs_set_eq:NN \ddots \@@_Ddots:
1617   \cs_set_eq:NN \iddots \@@_Iddots:
1618   \cs_set_eq:NN \dots \@@_Ldots:
1619   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1620 }

```

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.

```

1621 \hook_gput_code:nnn { begindocument } { . }
1622 {
1623   \IfPackageLoadedTF { booktabs }
1624   {
1625     \cs_new_protected:Npn \@@_patch_booktabs:
1626     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1627   }
1628   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1629 }

```

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`.

```

1630 \cs_new_protected:Npn \@@_some_initialization:
1631 {
1632   \@@_everycr:
1633   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1634   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1635   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1636   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1637   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1638   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1639 }

```

`\@@_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`.

```

1640 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1641 {

```

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

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

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

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.

```
1642 \seq_gclear:N \g_@@_pos_of_blocks_seq
```

Idem for other sequences written on the aux file.

```
1643 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1644 \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.

```
1645 \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.

```
1646 \fp_gzero:N \g_@@_total_X_weight_fp
1647 \bool_gset_false:N \g_@@_V_of_X_bool

1648 \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1649 \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol

1650 \@@_patch_booktabs:
1651 \box_clear_new:N \l_@@_cell_box
1652 \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}`).

```
1653 \bool_if:NT \l_@@_small_bool
1654 {
1655     \def \arraystretch { 0.47 }
1656     \dim_set:Nn \arraycolsep { 1.45 pt }
```

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

```
1657 \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1658 }
```

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!

```
1659 \bool_if:NT \g_@@_create_cell_nodes_bool
1660 {
1661     \tl_put_right:Nn \@@_begin_of_row:
1662     {
1663         \pgfsys@markposition
1664         { \@@_env: - row - \int_use:N \c@iRow - base }
1665     }
1666     \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1667 }
```

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`.

```
1668 \bool_if:NF \c_@@_revtex_bool
1669 {
1670     \def \ar@ialign
1671     {
1672         \tbl_init_cell_data_for_table:
1673         \@@_some_initialization:
1674         \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`.

```

1675         \cs_set_eq:Nc \ar@ialign { @@_old_ar@ialign: }
1676         \halign
1677     }
1678 }

```

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).

```

1679     \bool_if:NT \c_@@_revtex_bool
1680     {
1681         \IfPackageLoadedT { colortbl }
1682         { \cs_set_protected:Npn \CT@setup { } }
1683     }

```

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).

```

1684     \cs_set_eq:NN @@_old_ldots: \ldots
1685     \cs_set_eq:NN @@_old_cdots: \cdots
1686     \cs_set_eq:NN @@_old_vdots: \vdots
1687     \cs_set_eq:NN @@_old_ddots: \ddots
1688     \cs_set_eq:NN @@_old_iddots: \iddots
1689     \bool_if:NTF \l_@@_standard_cline_bool
1690     { \cs_set_eq:NN \cline @@_standard_cline: }
1691     { \cs_set_eq:NN \cline @@_cline: }
1692     \cs_set_eq:NN \Ldots @@_Ldots:
1693     \cs_set_eq:NN \Cdots @@_Cdots:
1694     \cs_set_eq:NN \Vdots @@_Vdots:
1695     \cs_set_eq:NN \Ddots @@_Ddots:
1696     \cs_set_eq:NN \Iddots @@_Iddots:
1697     \cs_set_eq:NN \Hline @@_Hline:
1698     \cs_set_eq:NN \Hspace @@_Hspace:
1699     \cs_set_eq:NN \Hdotsfor @@_Hdotsfor:
1700     \cs_set_eq:NN \Vdotsfor @@_Vdotsfor:
1701     \cs_set_eq:NN \Block @@_Block:
1702     \cs_set_eq:NN \rotate @@_rotate:
1703     \cs_set_eq:NN \OnlyMainNiceMatrix @@_OnlyMainNiceMatrix:n
1704     \cs_set_eq:NN \dotfill @@_dotfill:
1705     \cs_set_eq:NN \CodeAfter @@_CodeAfter:
1706     \cs_if_free:NT \Body { \cs_set_eq:NN \Body @@_Body: }
1707     \cs_set_eq:NN \diagbox @@_diagbox:nn
1708     \cs_set_eq:NN \NotEmpty @@_NotEmpty:
1709     \cs_set_eq:NN \TopRule @@_TopRule
1710     \cs_set_eq:NN \MidRule @@_MidRule
1711     \cs_set_eq:NN \BottomRule @@_BottomRule
1712     \cs_set_eq:NN \RowStyle @@_RowStyle:n
1713     \cs_set_eq:NN \Hbrace @@_Hbrace
1714     \cs_set_eq:NN \Vbrace @@_Vbrace
1715     \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1716     { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1717     \cs_set_eq:NN \cellcolor @@_cellcolor_tabular
1718     \cs_set_eq:NN \rowcolor @@_rowcolor_tabular
1719     \cs_set_eq:NN \rowcolors @@_rowcolors_tabular
1720     \cs_set_eq:NN \rowlistcolors @@_rowlistcolors_tabular
1721     \int_compare:nNnT { \l_@@_first_row_int } > { \c_zero_int }
1722     { \cs_set_eq:NN @@_tuning_first_row: \prg_do_nothing: }
1723     \int_compare:nNnT { \l_@@_last_row_int } < { \c_zero_int }
1724     { \cs_set_eq:NN @@_tuning_last_row: \prg_do_nothing: }

```

```
1725 \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:.`

```
1726 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1727 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1728 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1729 \@@_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!).

```
1730 \tl_if_exist:NT \l_@@_note_in_caption_tl
1731 {
1732   \tl_if_empty:NF \l_@@_note_in_caption_tl
1733   {
1734     \int_gset:Nn \g_@@_notes_caption_int { \l_@@_note_in_caption_tl }
1735     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1736   }
1737 }
```

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).

```
1738 \seq_gclear:N \g_@@_multicolumn_cells_seq
1739 \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).

```
1740 \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`).

```
1741 \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.

```
1742 \int_gzero:N \g_@@_col_total_int
1743 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1744 \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.

```
1745 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1746 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1747 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1748 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1749 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1750 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl
```

```
1751 \tl_gclear:N \g_nicematrix_code_before_tl
1752 \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]`).

```

1753 \dim_zero_new:N \l_@@_left_delim_dim
1754 \dim_zero_new:N \l_@@_right_delim_dim
1755 \bool_if:NTF \g_@@_delims_bool
1756 {

```

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

```

1757 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1758 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1759 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1760 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1761 }
1762 {
1763 \dim_gset:Nn \l_@@_left_delim_dim
1764 { 2 \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1765 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1766 }
1767 }

```

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.

```

1768 \cs_new_protected:Npn \@@_pre_array:
1769 {
1770 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1771 \int_gzero_new:N \c@iRow
1772 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1773 \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).

```

1774 \int_compare:nNnT \l_@@_last_row_int > 0
1775 { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1776 \int_compare:nNnT \l_@@_last_col_int > 0
1777 { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1778 \bool_if:NT \g_@@_aux_found_bool
1779 {
1780 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1781 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1782 \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1783 \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1784 }

```

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).

```

1785 \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
1786 {
1787 \bool_set_true:N \l_@@_last_row_without_value_bool
1788 \bool_if:NT \g_@@_aux_found_bool
1789 { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1790 }
1791 \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
1792 {
1793 \bool_if:NT \g_@@_aux_found_bool

```

```

1794     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1795 }

```

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”.

```

1796 \int_compare:nNtT { \l_@@_last_row_int } > { -2 }
1797 {
1798   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1799   {
1800     \dim_compare:nNtT { \g_@@_ht_last_row_dim } < { \box_ht:N \l_@@_cell_box }
1801     { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1802     \dim_compare:nNtT { \g_@@_dp_last_row_dim } < { \box_dp:N \l_@@_cell_box }
1803     { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1804   }
1805 }

1806 \seq_gclear:N \g_@@_cols_vlism_seq
1807 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1808 \bool_if:NT \l_@@_code_before_bool { \@@_exec_code_before: }

```

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

```

1809 \@@_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).

```

1810 \hbox_set:Nw \l_@@_the_array_box
1811 \skip_horizontal:N \l_@@_left_margin_dim
1812 \skip_horizontal:N \l_@@_extra_left_margin_dim
1813 \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).

```

1814 \m@th
1815 $ % $
1816 \bool_if:NtF \l_@@_light_syntax_bool
1817 { \use:c { @@-light-syntax } }
1818 { \use:c { @@-normal-syntax } }
1819 }

```

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

```

1820 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1821 {
1822   \tl_set:Nn \l_tmpa_tl { #1 }
1823   \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1824   { \@@_rescan_for_spanish:N \l_tmpa_tl }
1825   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1826   \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...

```

1827 \@@_pre_array:
1828 }

```

9 The \CodeBefore

```
1829 \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).

```
1830 \cs_new_protected:Npn \@@_pre_code_before:
1831 {
```

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.

```
1832 \pgfsys@markposition { \@@_env: - position }
1833 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1834 \pgfpicture
1835 \pgf@relevantforpicturesizefalse
```

First, the recreation of the row nodes.

```
1836 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int + 1 }
1837 {
1838   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1839   \pgfcoordinate { \@@_env: - row - ##1 }
1840   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1841 }
```

Now, the recreation of the col nodes.

```
1842 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int + 1 }
1843 {
1844   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1845   \pgfcoordinate { \@@_env: - col - ##1 }
1846   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1847 }
```

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

```
1848 \bool_if:NT \g_@@_create_cell_nodes_bool { \@@_recreate_cell_nodes: }
1849 \endpgfpicture
```

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

```
1850 \@@_create_diag_nodes:
```

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

```
1851 \@@_create_blocks_nodes:
1852 \IfPackageLoadedT { tikz }
1853 {
1854   \tikzset
1855   {
1856     every-picture / .style =
1857     { overlay , name~prefix = \@@_env: - }
1858   }
1859 }
1860 \cs_set_eq:NN \cellcolor \@@_cellcolor
1861 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1862 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1863 \cs_set_eq:NN \rowcolor \@@_rowcolor
1864 \cs_set_eq:NN \rowcolors \@@_rowcolors
1865 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1866 \cs_set_eq:NN \arraycolor \@@_arraycolor
1867 \cs_set_eq:NN \columncolor \@@_columncolor
1868 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1869 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1870 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1871 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1872 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1873 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1874 }
```

```

1875 \cs_new_protected:Npn \@@_exec_code_before:
1876 {

```

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...

```

1877 \clist_map_inline:Nn \l_@@_corners_cells_clist
1878 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1879 \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`.

```

1880 \@@_add_to_colors_seq:nn { { nocolor } } { }
1881 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1882 \group_begin:

```

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

```

1883 \if_mode_math:
1884 \@@_exec_code_before_i:
1885 \else:
1886 $ % $
1887 \@@_exec_code_before_i:
1888 $ % $
1889 \fi:
1890 \group_end:
1891 }

```

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`).

```

1892 \cs_new_protected:Npn \@@_exec_code_before_i:
1893 {
1894 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1895 { \@@_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`.

```

1896 \exp_last_unbraced:No \@@_CodeBefore_keys:
1897 \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.

```

1898 \@@_actually_color:
1899 \l_@@_code_before_tl
1900 \q_stop
1901 }

```

```

1902 \keys_define:nn { nicematrix / CodeBefore }
1903 {
1904 create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1905 create-cell-nodes .default:n = true ,
1906 sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1907 sub-matrix .value_required:n = true ,
1908 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1909 delimiters / color .value_required:n = true ,
1910 unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1911 }
1912 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1913 {

```

```

1914 \keys_set:nn { nicematrix / CodeBefore } { #1 }
1915 \@@_CodeBefore:w
1916 }

```

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.

```

1917 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1918 {
1919   \bool_if:NTF \g_@@_aux_found_bool
1920   {
1921     \@@_pre_code_before:
1922     \legacy_if:nF { measuring@ } { #1 }
1923   }

```

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).

```

1924 {
1925   \pgfsys@markposition { \@@_env: - position }
1926   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1927   \pgfpicture
1928   \pgf@relevantforpicturesizefalse
1929   \endpgfpicture

```

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

```

1930   \pgfpicture
1931   \pgfrememberpicturepositiononpagetrue
1932   \endpgfpicture

```

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

```

1933   \pgfpicture
1934   \pgf@relevantforpicturesizefalse
1935   \pgfrememberpicturepositiononpagetrue
1936   \endpgfpicture

```

The following picture corresponds `\@@_actually_color:`

```

1937   \pgfpicture
1938   \pgf@relevantforpicturesizefalse
1939   \endpgfpicture
1940 }
1941 }

```

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.

```

1942 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1943 {
1944   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
1945   {
1946     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1947     \pgfcoordinate { \@@_env: - row - ##1 - base }
1948     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1949     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
1950     {
1951       \cs_if_exist:cT
1952       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1953       {
1954         \pgfsys@getposition
1955         { \@@_env: - ##1 - #####1 - NW }
1956         \@@_node_position:
1957         \pgfsys@getposition
1958         { \@@_env: - ##1 - #####1 - SE }
1959         \@@_node_position_i:

```



```

1960         \@@_pgf_rect_node:nnn
1961         { \@@_env: - ##1 - ###1 }
1962         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1963         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1964     }
1965 }
1966 }
1967 \@@_create_extra_nodes:
1968 \@@_create_aliases_last:
1969 }

1970 \cs_new_protected:Npn \@@_create_aliases_last:
1971 {
1972     \int_step_inline:nn { \c@iRow }
1973     {
1974         \pgfnodealias
1975         { \@@_env: - ##1 - last }
1976         { \@@_env: - ##1 - \int_use:N \c@jCol }
1977     }
1978     \int_step_inline:nn { \c@jCol }
1979     {
1980         \pgfnodealias
1981         { \@@_env: - last - ##1 }
1982         { \@@_env: - \int_use:N \c@iRow - ##1 }
1983     }
1984     \pgfnodealias
1985     { \@@_env: - last - last }
1986     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1987 }

1988 \cs_new_protected:Npn \@@_create_blocks_nodes:
1989 {
1990     \pgfpicture
1991     \pgf@relevantforpicturesizefalse
1992     \pgfrememberpicturepositiononpagetrue
1993     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1994     { \@@_create_one_block_node:nnnnn ##1 }
1995     \endpgfpicture
1996 }

```

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

```

1997 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1998 {
1999     \tl_if_empty:nF { #5 }
2000     {
2001         \@@_qpoint:n { col - #2 }
2002         \dim_set_eq:NN \l_tmpa_dim \pgf@x
2003         \@@_qpoint:n { #1 }
2004         \dim_set_eq:NN \l_tmpb_dim \pgf@y
2005         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
2006         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
2007         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
2008         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
2009         \@@_pgf_rect_node:nnnnn
2010         { \@@_env: - #5 }
2011         { \dim_use:N \l_tmpa_dim }
2012         { \dim_use:N \l_tmpb_dim }
2013         { \dim_use:N \l_@@_tmpc_dim }

```

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).

```

2014         { \dim_use:N \l_@@_tmpd_dim }
2015     }
2016 }

2017 \cs_new_protected:Npn \@@_patch_for_revtext:
2018 {
2019     \cs_set_eq:NN \@addamp \@addamp@LaTeX
2020     \cs_set_eq:NN \@array \@array@array
2021     \cs_set_eq:NN \@tabular \@tabular@array
2022     \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
2023     \cs_set_eq:NN \array \array@array
2024     \cs_set_eq:NN \endarray \endarray@array
2025     \cs_set:Npn \endtabular { \endarray $\egroup } % $
2026     \cs_set_eq:NN \@mkpream \@mkpream@array
2027     \cs_set_eq:NN \@classx \@classx@array
2028     \cs_set_eq:NN \insert@column \insert@column@array
2029     \cs_set_eq:NN \@arraycr \@arraycr@array
2030     \cs_set_eq:NN \@xarraycr \@xarraycr@array
2031     \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
2032 }

```

10 The environment {NiceArrayWithDelims}

```

2033 \NewDocumentEnvironment { NiceArrayWithDelims }
2034 { m m 0 { } m ! 0 { } t \CodeBefore }
2035 {
2036     \bool_if:NT \c_@@_revtext_bool { \@@_patch_for_revtext: }
2037
2038     \@@_provide_pgfsyspdfmark:
2039     \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.

```

2039     \bgroup

2040     \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2041     \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2042     \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
2043     \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

2044     \int_gzero:N \g_@@_block_box_int
2045     \dim_gzero:N \g_@@_width_last_col_dim
2046     \dim_gzero:N \g_@@_width_first_col_dim
2047     \bool_gset_false:N \g_@@_row_of_col_done_bool
2048     \str_if_empty:NT \g_@@_name_env_str
2049     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
2050     \bool_if:NTF \l_@@_tabular_bool
2051     { \mode_leave_vertical: }
2052     { \@@_test_if_math_mode: }
2053     \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
2054     \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.

```

2055     \cs_gset_eq:cN { @@_old_CT@arc@ } \CT@arc@

```

⁸e.g. `\color[rgb]{0.5,0.5,0}`

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

```

2056 \cs_if_exist:NT \tikz@library@external@loaded
2057 {
2058     \tikzexternaldisable
2059     \cs_if_exist:NT \ifstandalone
2060     { \tikzset { external / optimize = false } }
2061 }

```

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

```

2062 \int_gincr:N \g_@@_env_int
2063 \bool_if:NF \l_@@_block_auto_columns_width_bool
2064 { \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.

```

2065 \seq_gclear:N \g_@@_blocks_seq
2066 \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`.

```

2067 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
2068 \seq_gclear:N \g_@@_pos_of_xdots_seq
2069 \tl_gclear_new:N \g_@@_code_before_tl
2070 \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.

```

2071 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2072 {
2073     \bool_gset_true:N \g_@@_aux_found_bool
2074     \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2075 }
2076 { \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.

```

2077 \tl_gclear:N \g_@@_aux_tl
2078 \tl_if_empty:NF \g_@@_code_before_tl
2079 {
2080     \bool_set_true:N \l_@@_code_before_bool
2081     \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
2082 }
2083 \tl_if_empty:NF \g_@@_pre_code_before_tl
2084 { \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`.

```

2085 \bool_if:NTF \g_@@_delims_bool
2086 { \keys_set:nn { nicematrix / pNiceArray } }
2087 { \keys_set:nn { nicematrix / NiceArray } }
2088 { #3 , #5 }

2089 \@@_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:.`

```

2090 \bool_if:nTF { #6 } { \@@_CodeBefore_Body:w } { \@@_pre_array: }

```

2091 }

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

```
2092 {
2093   \bool_if:NTF \l_@@_light_syntax_bool
2094     { \use:c { end @@-light-syntax } }
2095     { \use:c { end @@-normal-syntax } }
2096   $ % $
2097   \skip_horizontal:N \l_@@_right_margin_dim
2098   \skip_horizontal:N \l_@@_extra_right_margin_dim
2099   \hbox_set_end:
2100   \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.

```
2101   \bool_if:NT \l_@@_width_used_bool
2102   {
2103     \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2104       { \@@_error_or_warning:n { width~without~X~columns } }
2105   }
```

Now, if there is at least one `X`-column in the environment, we compute the width that those columns will have (in the next compilation). In fact, `\l_@@_X_columns_dim` will be the width of a column of weight 1.0. For a `X`-column of weight x , the width will be `\l_@@_X_columns_dim` multiplied by x .

```
2106   \fp_compare:nNnT { \g_@@_total_X_weight_fp } > { \c_zero_fp }
2107     { \@@_compute_width_X: }
```

If the user has used the key `last-row` with a value, we control that the given value is correct (since we have just constructed the array, we know the actual number of rows of the array).

```
2108   \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
2109   {
2110     \bool_if:NF \l_@@_last_row_without_value_bool
2111     {
2112       \int_compare:nNnF { \l_@@_last_row_int } = { \c@iRow }
2113       {
2114         \@@_error:n { Wrong~last~row }
2115         \int_set_eq:NN \l_@@_last_row_int \c@iRow
2116       }
2117     }
2118   }
```

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

```
2119   \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2120   \bool_if:NTF \g_@@_last_col_found_bool
2121     { \int_gdecr:N \c@jCol }
2122     {
2123       \int_compare:nNnT { \l_@@_last_col_int } > { -1 }
2124       { \@@_error:n { last~col~not~used } }
2125     }
```

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

```
2126   \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2127   \int_compare:nNnT { \l_@@_last_row_int } > { -1 }
2128     { \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. 95).

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

```

2129 \int_if_zero:nT { \l_@@_first_col_int }
2130 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2131 \bool_if:nTF { ! \g_@@_delims_bool }
2132 {
2133   \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
2134   { \@@_use_arraybox_with_notes_c: }
2135   {
2136     \str_if_eq:eeTF { \l_@@_baseline_tl } { b }
2137     { \@@_use_arraybox_with_notes_b: }
2138     { \@@_use_arraybox_with_notes: }
2139   }
2140 }

```

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).

```

2141 {
2142   \int_if_zero:nTF { \l_@@_first_row_int }
2143   {
2144     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2145     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2146   }
2147   { \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

```

2148 \int_compare:nNnTF { \l_@@_last_row_int } > { -2 }
2149 {
2150   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2151   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2152 }
2153 { \dim_zero:N \l_tmpb_dim }
2154 \hbox_set:Nn \l_tmpa_box
2155 {
2156   \m@th
2157   $ % $
2158   \@@_color:o \l_@@_delimiters_color_tl
2159   \exp_after:wN \left \g_@@_left_delim_tl
2160   \vcenter
2161   {

```

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.

```

2162   \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2163   \hbox
2164   {
2165     \bool_if:NTF \l_@@_tabular_bool
2166     { \skip_horizontal:n { - \tabcolsep } }
2167     { \skip_horizontal:n { - \arraycolsep } }
2168     \@@_use_arraybox_with_notes_c:
2169     \bool_if:NTF \l_@@_tabular_bool
2170     { \skip_horizontal:n { - \tabcolsep } }
2171     { \skip_horizontal:n { - \arraycolsep } }
2172   }

```

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

```

2173   \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2174   }
2175   \exp_after:wN \right \g_@@_right_delim_tl
2176   $ % $
2177 }

```

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).

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.

```

2178     \bool_if:NTF \l_@@_delimiters_max_width_bool
2179     {
2180         \@@_put_box_in_flow_bis:nn
2181         { \g_@@_left_delim_tl }
2182         { \g_@@_right_delim_tl }
2183     }
2184     \@@_put_box_in_flow:
2185 }

```

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. 96).

```

2186     \bool_if:NT \g_@@_last_col_found_bool
2187     { \skip_horizontal:N \g_@@_width_last_col_dim }
2188     \bool_if:NT \l_@@_preamble_bool
2189     {
2190         \int_compare:nNnT { \c@jCol } < { \g_@@_static_num_of_col_int }
2191         { \@@_err_columns_not_used: }
2192     }
2193     \@@_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.

```

2194     \egroup

```

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

```

2195     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2196     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2197     \iow_now:Ne \@mainaux
2198     {
2199         \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2200         \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2201         { \exp_not:o \g_@@_aux_tl }
2202     }
2203     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2204     \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2205 }

```

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

```

2206 \cs_new_protected:Npn \@@_err_columns_not_used:
2207 {
2208     \@@_warning:n { columns~not~used }
2209     \cs_gset:Npn \@@_err_columns_not_used: { }
2210 }

```

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 .

```

2211 \cs_new_protected:Npn \@@_compute_width_X:
2212 {
2213     \tl_gput_right:Ne \g_@@_aux_tl
2214     {
2215         \bool_set_true:N \l_@@_X_columns_aux_bool
2216         \dim_set:Nn \l_@@_X_columns_dim
2217         {

```

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`

```

2218 \bool_lazy_and:nnTF
2219 { \g_@@_V_of_X_bool }
2220 { \l_@@_X_columns_aux_bool }
2221 { \dim_use:N \l_@@_X_columns_dim }
2222 {
2223   \dim_compare:nNnTF
2224   {
2225     \dim_abs:n
2226     { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2227   }
2228   <
2229   { 0.001 pt }
2230   { \dim_use:N \l_@@_X_columns_dim }
2231   {
2232     \dim_eval:n
2233     {
2234       \l_@@_X_columns_dim
2235       +
2236       \fp_to_dim:n
2237       {
2238         (
2239           \dim_eval:n
2240           { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2241         )
2242         / \fp_use:N \g_@@_total_X_weight_fp
2243       }
2244     }
2245   }
2246 }
2247 }
2248 }
2249 }

```

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`.

```

2250 \cs_new_protected:Npn \@@_transform_preamble:
2251 {
2252   \@@_transform_preamble_i:
2253   \@@_transform_preamble_ii:
2254 }
2255 \cs_new_protected:Npn \@@_transform_preamble_i:
2256 {
2257   \int_gzero:N \c@jCol

```

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

```

2258 \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.

```

2259 \bool_gset_false:N \g_tmpb_bool

```

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

```

2260 \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2261 \int_zero:N \l_tmpa_int
2262 \tl_gclear:N \g_@@_array_preamble_tl
2263 \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2264 {
2265   \tl_gset:Nn \g_@@_array_preamble_tl
2266     { ! { \skip_horizontal:N \arrayrulewidth } }
2267 }
2268 {
2269   \clist_if_in:NnT \l_@@_vlines_clist 1
2270   {
2271     \tl_gset:Nn \g_@@_array_preamble_tl
2272       { ! { \skip_horizontal:N \arrayrulewidth } }
2273   }
2274 }

```

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

```

2275 \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2276 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2277 \@@_replace_columncolor:
2278 }

2279 \cs_new_protected:Npn \@@_transform_preamble_ii:
2280 {

```

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

```

2281 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2282 {
2283   \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2284   { \bool_gset_true:N \g_@@_delims_bool }
2285 }
2286 { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2287 \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).

```

2288 \int_if_zero:nTF { \l_@@_first_col_int }
2289 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2290 {
2291   \bool_if:NF \g_@@_delims_bool
2292   {
2293     \bool_if:NF \l_@@_tabular_bool
2294     {
2295       \clist_if_empty:NT \l_@@_vlines_clist
2296       {
2297         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2298         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2299       }
2300     }
2301   }
2302 }
2303 \int_compare:nNnTF { \l_@@_last_col_int } > { -1 }
2304 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2305 {
2306   \bool_if:NF \g_@@_delims_bool
2307   {

```



```

2308     \bool_if:NF \l_@@_tabular_bool
2309     {
2310         \clist_if_empty:NT \l_@@_vlines_clist
2311         {
2312             \bool_if:NF \l_@@_exterior_arraycolsep_bool
2313             { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2314         }
2315     }
2316 }
2317 }

```

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.

```

2318     \tag_if_active:F
2319     {

```

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

```

2320     \dim_compare:nNnT { \l_@@_tabular_width_dim } = { \c_zero_dim }
2321     {
2322         \tl_gput_right:Nn \g_@@_array_preamble_tl
2323         { > { \@@_err_too_many_cols: } 1 }
2324     }
2325 }
2326 }

```

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.

```

2327 \cs_new_protected:Npn \@@_rec_preamble:n #1
2328 {

```

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

```

2329     \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2330     { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2331     {

```

Now, the columns defined by \newcolumntype of array.

```

2332     \cs_if_exist:cTF { NC @ find @ #1 }
2333     {
2334         \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2335         \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2336     }
2337     {
2338         \str_if_eq:nnTF { #1 } { S }
2339         { \@@_fatal:n { unknown~column~type~S } }
2340         { \@@_fatal:nn { unknown~column~type } { #1 } }
2341     }
2342 }
2343 }

```

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 c, l and r

```

2344 \cs_new_protected:Npn \@@_c: #1
2345 {
2346   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2347   \tl_gclear:N \g_@@_pre_cell_tl
2348   \tl_gput_right:Nn \g_@@_array_preamble_tl
2349     { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2350   \int_gincr:N \c@jCol
2351   \@@_rec_preamble_after_col:n
2352 }

2353 \cs_new_protected:Npn \@@_l: #1
2354 {
2355   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2356   \tl_gclear:N \g_@@_pre_cell_tl
2357   \tl_gput_right:Nn \g_@@_array_preamble_tl
2358     {
2359       > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2360       l
2361       < \@@_cell_end:
2362     }
2363   \int_gincr:N \c@jCol
2364   \@@_rec_preamble_after_col:n
2365 }

2366 \cs_new_protected:Npn \@@_r: #1
2367 {
2368   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2369   \tl_gclear:N \g_@@_pre_cell_tl
2370   \tl_gput_right:Nn \g_@@_array_preamble_tl
2371     {
2372       > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2373       r
2374       < \@@_cell_end:
2375     }
2376   \int_gincr:N \c@jCol
2377   \@@_rec_preamble_after_col:n
2378 }

```

For ! and @

```

2379 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2380 {
2381   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2382   \@@_rec_preamble:n
2383 }
2384 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2385 \cs_new_protected:cpn { @@ _ | : } #1
2386 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2387   \int_incr:N \l_tmpa_int
2388   \@@_make_preamble_i_i:n
2389 }

2390 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2391 {

```

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

```

2392   \str_if_eq:nnTF { #1 } { | }
2393     { \use:c { @@ _ | : } | }
2394     { \@@_make_preamble_i_ii:nn { } #1 }
2395 }

```

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

```

2396 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2397 {
2398   \str_if_eq:nnTF { #2 } { [ ] }
2399   { \@@_make_preamble_i_ii:nw { #1 } [ ] }
2400   { \@@_make_preamble_i_iii:nn { #2 } { #1 } } }
2401 }
2402 \cs_new_protected:Npn \@@_make_preamble_i_iii:nw #1 [ #2 ]
2403 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2404 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2405 {
2406   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2407   \tl_gput_right:Ne \g_@@_array_preamble_tl
2408   {

```

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

```

2409   \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2410   }
2411   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2412   {
2413     \@@_vline:n
2414     {
2415       position = \int_eval:n { \c@jCol + 1 } ,
2416       multiplicity = \int_use:N \l_tmpa_int ,
2417       total-width = \dim_use:N \l_@@_rule_width_dim ,
2418       #2
2419     }

```

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.

```

2420   }
2421   \int_zero:N \l_tmpa_int
2422   \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2423   \@@_rec_preamble:n #1
2424   }
2425 \cs_new_protected:cpn { @@_ > : } #1 #2
2426 {
2427   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2428   \@@_rec_preamble:n
2429   }
2430 \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.

```

2431 \keys_define:nn { nicematrix / p-column }
2432 {
2433   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2434   r .value_forbidden:n = true ,
2435   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2436   c .value_forbidden:n = true ,
2437   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2438   l .value_forbidden:n = true ,
2439   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2440   S .value_forbidden:n = true ,
2441   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2442   p .value_forbidden:n = true ,
2443   t .meta:n = p ,
2444   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2445   m .value_forbidden:n = true ,

```

```

2446     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2447     b .value_forbidden:n = true
2448 }

```

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

```

2449 \cs_new_protected:Npn \@@_p: #1
2450 {
2451     \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2452     \@@_make_preamble_ii_i:n
2453 }
2454 \cs_new_eq:NN \@@_b: \@@_p:
2455 \cs_new_eq:NN \@@_m: \@@_p:
2456 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2457 {
2458     \str_if_eq:nnTF { #1 } { [ ] {
2459         { \@@_make_preamble_ii_ii:w [ ] }
2460         { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2461     }
2462     \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2463     { \@@_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.

```

2464 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2465 {

```

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).

```

2466     \str_set:Nn \l_@@_hpos_col_str { j }
2467     \@@_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`.

```

2468     \setlength { \l_tmpa_dim } { #2 }
2469     \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2470 }
2471 \cs_new_protected:Npn \@@_keys_p_column:n #1
2472 { \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.

```

2473 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2474 {

```

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

```

2475     \use:e
2476     {
2477         \@@_make_preamble_ii_vi:nnnnnnnn
2478         { \str_if_eq:eeTF { \l_@@_vpos_col_str } { p } { t } { b } }
2479         { #1 }
2480         {
2481             \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.

```

2482         \str_if_eq:eeTF { \l_@@_hpos_col_str } { j }
2483         { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2484     {

```

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

```

2485         \def \exp_not:N \l_@@_hpos_cell_tl
2486         { \str_lowercase:f { \l_@@_hpos_col_str } }
2487     }
2488     \IfPackageLoadedTF { ragged2e }
2489     {
2490         \str_case:on \l_@@_hpos_col_str
2491         {

```

The following `\exp_not:N` are mandatory.

```

2492             c { \exp_not:N \Centering }
2493             l { \exp_not:N \RaggedRight }
2494             r { \exp_not:N \RaggedLeft }
2495         }
2496     }
2497     {
2498         \str_case:on \l_@@_hpos_col_str
2499         {
2500             c { \exp_not:N \centering }
2501             l { \exp_not:N \raggedright }
2502             r { \exp_not:N \raggedleft }
2503         }
2504     }
2505     #3
2506 }
2507 { \str_if_eq:eeT { \l_@@_vpos_col_str } { m } \@@_center_cell_box: }
2508 { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_begin:w }
2509 { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_end: }
2510 { #2 }
2511 {
2512     \str_case:onF \l_@@_hpos_col_str
2513     {
2514         { j } { c }
2515         { si } { c }
2516     }

```

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

```

2517         { \str_lowercase:f \l_@@_hpos_col_str }
2518     }
2519 }

```

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

```

2520     \int_gincr:N \c@jCol
2521     \@@_rec_preamble_after_col:n
2522 }

```

#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*.

```

2523 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2524 {
2525   \str_if_eq:eeTF { \l_@@_hpos_col_str } { si }
2526   {
2527     \tl_gput_right:Nn \g_@@_array_preamble_tl
2528     { > \@@_test_if_empty_for_S: }
2529   }
2530   {
2531     \str_if_eq:eeTF { #7 } { varwidth }
2532     {
2533       \tl_gput_right:Nn \g_@@_array_preamble_tl
2534       { > \@@_test_if_empty_varwidth: }
2535     }
2536     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2537   }
2538   \tl_gput_right:Nn \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2539   \tl_gclear:N \g_@@_pre_cell_tl
2540   \tl_gput_right:Nn \g_@@_array_preamble_tl
2541   {
2542     > {

```

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.

```

2543       \dim_set:Nn \l_@@_col_width_dim { #2 }
2544       \@@_cell_begin:

```

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

```

2545       \use:c { #7 } [ #1 ] { #2 }

```

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

```

2546   \everypar
2547   {
2548     \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2549     \everypar { }
2550   }

```

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

```

2551       #3

```

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

```

2552       \g_@@_row_style_tl
2553       \arraybackslash
2554       #5
2555     }
2556     #8
2557     < {
2558       #6

```

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

```

2559       \@finalstrut \@arstrutbox
2560       \use:c { end #7 }

```

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

```

2561       #4
2562       \@@_cell_end:
2563     }
2564   }
2565 }

```

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

```
2566 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2567 {
```

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 `&`).

```
2568 \group_align_safe_begin:
2569 \peek_meaning:NTF &
2570 { \@@_the_cell_is_empty: }
2571 {
2572   \peek_meaning:NTF \\\
2573   { \@@_the_cell_is_empty: }
2574   {
2575     \peek_meaning:NTF \crcr
2576     \@@_the_cell_is_empty:
2577     \group_align_safe_end:
2578   }
2579 }
2580 }
```

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

```
2581 \cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
2582 {
2583   \group_align_safe_begin:
2584   \peek_meaning:NTF &
2585   { \@@_the_cell_is_empty_varwidth: }
2586   {
2587     \peek_meaning:NTF \\\
2588     { \@@_the_cell_is_empty_varwidth: }
2589     {
2590       \peek_meaning:NTF \crcr
2591       \@@_the_cell_is_empty_varwidth:
2592       \group_align_safe_end:
2593     }
2594   }
2595 }

2596 \cs_new_protected:Npn \@@_the_cell_is_empty:
2597 {
2598   \group_align_safe_end:
2599   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2600   {
```

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

```
2601   \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).

```
2602   \skip_horizontal:N \l_@@_col_width_dim
2603 }
2604 }

2605 \cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
2606 {
2607   \group_align_safe_end:
2608   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2609   { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2610 }
```

```

2611 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2612 {
2613   \peek_meaning:NT \__siunitx_table_skip:n
2614   { \bool_gset_true:N \g_@@_empty_cell_bool }
2615 }

```

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 than the height of `\strutbox`, there is only one row.

```

2616 \cs_new_protected:Npn \@@_center_cell_box:
2617 {

```

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

```

2618   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2619   {
2620     \dim_compare:nNnT
2621     { \box_ht:N \l_@@_cell_box }
2622     >

```

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).

```

2623     { \box_ht:N \strutbox }
2624     {
2625       \hbox_set:Nn \l_@@_cell_box
2626       {
2627         \box_move_down:nn
2628         {
2629           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2630             + \baselineskip ) / 2
2631         }
2632         { \box_use:N \l_@@_cell_box }
2633       }
2634     }
2635   }
2636 }

```

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

```

2637 \cs_new_protected:Npn \@@_V: #1 #2
2638 {
2639   \str_if_eq:nnTF { #2 } { [ ] }
2640   { \@@_make_preamble_V_i:w [ ] }
2641   { \@@_make_preamble_V_i:w [ ] { #2 } }
2642 }
2643 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2644 { \@@_make_preamble_V_ii:nn { #1 } }
2645 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2646 {
2647   \str_set:Nn \l_@@_vpos_col_str { p }
2648   \str_set:Nn \l_@@_hpos_col_str { j }
2649   \@@_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`.

```

2650   \setlength { \l_tmpa_dim } { #2 }
2651   \IfPackageLoadedTF { varwidth }
2652   { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2653   {

```



```

2654     \@@_error_or_warning:n { varwidth-not-loaded }
2655     \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2656   }
2657 }

```

For `w` and `W`

```

2658 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2659 \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.

```

2660 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2661 {
2662   \str_if_eq:nnTF { #3 } { s }
2663     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2664     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2665 }

```

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.

```

2666 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2667 {
2668   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2669   \tl_gclear:N \g_@@_pre_cell_tl
2670   \tl_gput_right:Nn \g_@@_array_preamble_tl
2671     {
2672       > {

```

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`.

```

2673     \setlength { \l_@@_col_width_dim } { #2 }
2674     \@@_cell_begin:
2675     \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2676   }
2677   c
2678   < {
2679     \@@_cell_end_for_w_s:
2680     #1
2681     \@@_adjust_size_box:
2682     \box_use_drop:N \l_@@_cell_box
2683   }
2684 }
2685 \int_gincr:N \c@jCol
2686 \@@_rec_preamble_after_col:n
2687 }

```

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

```

2688 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2689 {
2690   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2691   \tl_gclear:N \g_@@_pre_cell_tl
2692   \tl_gput_right:Nn \g_@@_array_preamble_tl
2693     {
2694       > {

```

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`.

```

2695         \setlength { \l_@@_col_width_dim } { #4 }
2696         \hbox_set:Nw \l_@@_cell_box
2697         \@@_cell_begin:
2698         \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2699     }
2700     c
2701     < {
2702         \@@_cell_end:
2703         \hbox_set_end:
2704         #1
2705         \@@_adjust_size_box:
2706         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2707     }
2708 }
```

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

```

2709     \int_gincr:N \c@jCol
2710     \@@_rec_preamble_after_col:n
2711 }

2712 \cs_new_protected:Npn \@@_special_W:
2713 {
2714     \dim_compare:nNt { \box_wd:N \l_@@_cell_box } > { \l_@@_col_width_dim }
2715     { \@@_warning:n { W~warning } }
2716 }
```

For `S` (of `siunitx`).

```

2717 \cs_new_protected:Npn \@@_S: #1 #2
2718 {
2719     \str_if_eq:nnTF { #2 } { [ ] }
2720     { \@@_make_preamble_S:w [ ] }
2721     { \@@_make_preamble_S:w [ ] { #2 } }
2722 }

2723 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2724 { \@@_make_preamble_S_i:n { #1 } }

2725 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2726 {
2727     \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2728     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2729     \tl_gclear:N \g_@@_pre_cell_tl
2730     \tl_gput_right:Nn \g_@@_array_preamble_tl
2731     {
2732         > {
```

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).

```

2733         \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2734         \keys_set:nn { siunitx } { #1 }
2735         \@@_cell_begin:
2736         \siunitx_cell_begin:w
2737     }
2738     c
2739     <
2740     {
2741         \siunitx_cell_end:
```

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

```

2742         \tl_gput_right:Nx \g_@@_cell_after_hook_tl
2743         {
2744             \bool_if:NTF \l__siunitx_table_text_bool
2745             { \bool_set_true:N }
2746             { \bool_set_false:N }
2747             \l__siunitx_table_text_bool
2748         }
2749         \@@_cell_end:
2750     }
2751 }

```

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

```

2752     \int_gincr:N \c@jCol
2753     \@@_rec_preamble_after_col:n
2754 }

```

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

```

2755 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2756 {
2757     \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.

```

2758     \int_if_zero:nTF { \c@jCol }
2759     {
2760         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2761         {

```

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

```

2762         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2763         \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2764         \@@_rec_preamble:n #2
2765     }
2766     {
2767         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2768         \@@_make_preamble_iv:nn { #1 } { #2 }
2769     }
2770 }
2771 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2772 }
2773 \cs_set_eq:cc { @@ _ \token_to_str:N [ : } { @@ _ \token_to_str:N ( : }
2774 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2775 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2776 {
2777     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2778     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2779     \tl_if_in:nnTF { ( [ \{ ) } \} \left \right } { #2 }
2780     {
2781         \@@_error:nn { delimiter~after~opening } { #2 }
2782         \@@_rec_preamble:n
2783     }
2784     { \@@_rec_preamble:n #2 }
2785 }

```

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

```

2786 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2787 { \use:c { @@ _ \token_to_str:N ( : } }

```

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

```

2788 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2789 {
2790   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2791   \tl_if_in:nnTF { ) ] \} } { #2 }
2792   { \@@_make_preamble_v:nnn #1 #2 }
2793   {
2794     \str_if_eq:nnTF { \s_stop } { #2 }
2795     {
2796       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2797       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2798       {
2799         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2800         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2801         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2802         \@@_rec_preamble:n #2
2803       }
2804     }
2805     {
2806       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2807       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2808       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2809       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2810       \@@_rec_preamble:n #2
2811     }
2812   }
2813 }
2814 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2815 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2816 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2817 {
2818   \str_if_eq:nnTF { \s_stop } { #3 }
2819   {
2820     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2821     {
2822       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2823       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2824       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2825       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2826     }
2827     {
2828       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2829       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2830       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2831       \@@_error:nn { double~closing~delimiter } { #2 }
2832     }
2833   }
2834   {
2835     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2836     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2837     \@@_error:nn { double~closing~delimiter } { #2 }
2838     \@@_rec_preamble:n #3
2839   }
2840 }
2841 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2842 { \use:c { @@ _ \token_to_str:N ) : } }

```

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

potential <{...}, we have to insert !{\skip_horizontal:N ...} when the key vlines is used. In fact, we have also to test whether there is, after the <{...}, a @{...}.

```

2843 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2844 {
2845   \str_if_eq:nnTF { #1 } { < }
2846   { \@@_rec_preamble_after_col_i:n }
2847   {
2848     \str_if_eq:nnTF { #1 } { @ }
2849     { \@@_rec_preamble_after_col_ii:n }
2850     {
2851       \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2852       {
2853         \tl_gput_right:Nn \g_@@_array_preamble_tl
2854         { ! { \skip_horizontal:N \arrayrulewidth } }
2855       }
2856       {
2857         \clist_if_in:NcT \l_@@_vlines_clist
2858         { \int_eval:n { \c@jCol + 1 } }
2859         {
2860           \tl_gput_right:Nn \g_@@_array_preamble_tl
2861           { ! { \skip_horizontal:N \arrayrulewidth } }
2862         }
2863       }
2864       \@@_rec_preamble:n { #1 }
2865     }
2866   }
2867 }

2868 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2869 {
2870   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2871   \@@_rec_preamble_after_col:n
2872 }

```

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

```

2873 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2874 {
2875   \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2876   {
2877     \tl_gput_right:Nn \g_@@_array_preamble_tl
2878     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2879   }
2880   {
2881     \clist_if_in:NcTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2882     {
2883       \tl_gput_right:Nn \g_@@_array_preamble_tl
2884       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2885     }
2886     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2887   }
2888   \@@_rec_preamble:n
2889 }

2890 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2891 {
2892   \tl_clear:N \l_tmpa_tl
2893   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2894   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2895 }

```

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

```

2896 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2897 { \@@_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.

```

2898 \cs_new_protected:Npn \@@_X: #1 #2
2899 {
2900   \str_if_eq:nnTF { #2 } { [ ]
2901     { \@@_make_preamble_X:w [ ]
2902       { \@@_make_preamble_X:w [ ] #2 }
2903     }
2904   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2905   { \@@_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`.

```

2906 \keys_define:nn { nicematrix / X-column }
2907 {
2908   V .code:n =
2909   \IfPackageLoadedTF { varwidth }
2910   {
2911     \bool_set_true:N \l_@@_V_of_X_bool
2912     \bool_gset_true:N \g_@@_V_of_X_bool
2913   }
2914   { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
2915   unknown .code:n =
2916   \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2917   { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2918   { \@@_error_or_warning:n { invalid~weight } }
2919 }

```

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

```

2920 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2921 {

```

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).

```

2922   \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).

```

2923   \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`).

```

2924   \fp_set:Nn \l_tmpa_fp { 1.0 }
2925   \@@_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.

```

2926   \bool_set_false:N \l_@@_V_of_X_bool
2927   \keys_set:no { nicematrix / X-column } \l_tmpa_tl

```

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

```

2928   \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).

```
2929 \bool_if:NTF \l_@@_X_columns_aux_bool
2930 {
2931 \@@_make_preamble_ii_iv:nnn
```

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

```
2932 { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2933 { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
2934 { \@@_no_update_width: }
2935 }
```

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.

```
2936 {
2937 \tl_gput_right:Nn \g_@@_array_preamble_tl
2938 {
2939 > {
2940 \@@_cell_begin:
2941 \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`.

```
2942 \NotEmpty
```

The following code will nullify the box of the cell.

```
2943 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2944 { \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`.

```
2945 \begin { minipage } { 5 cm } \arraybackslash
2946 }
2947 c
2948 < {
2949 \end { minipage }
2950 \@@_cell_end:
2951 }
2952 }
2953 \int_gincr:N \c@jCol
2954 \@@_rec_preamble_after_col:n
2955 }
2956 }
```

```
2957 \cs_new_protected:Npn \@@_no_update_width:
2958 {
2959 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2960 { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2961 }
```

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

```
2962 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2963 {
2964 \seq_gput_right:Ne \g_@@_cols_vlism_seq
2965 { \int_eval:n { \c@jCol + 1 } }
2966 \tl_gput_right:Ne \g_@@_array_preamble_tl
2967 { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2968 \@@_rec_preamble:n
2969 }
```

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.

```
2970 \cs_set_eq:cn { @@ _ \token_to_str:N \s_stop : } \use_none:n
```

The following lines try to catch some errors (when the final user has forgotten the preamble of its environment).

```
2971 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2972 { \@@_fatal:n { Preamble-forgotten } }
2973 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
2974 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
2975 { @@ _ \token_to_str:N \hline : }
2976 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
2977 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
2978 { @@ _ \token_to_str:N \hline : }
2979 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
2980 { @@ _ \token_to_str:N \hline : }
2981 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
2982 { @@ _ \token_to_str:N \hline : }
2983 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
2984 { @@ _ \token_to_str:N \hline : }
```

12 The redefinition of `\multicolumn`

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

```
2985 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2986 {
```

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

```
2987 \multispan { #1 }
2988 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2989 \begingroup
2990 \tbl_update_multicolumn_cell_data:n { #1 }
```

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

```
2991 \tl_gclear:N \g_@@_preamble_tl
2992 \@@_make_m_preamble:n #2 \q_stop
```

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

```
2993 \def \@addamp
2994 {
2995 \legacy_if:nTF { @firstamp }
2996 { \legacy_if_set_false:n { @firstamp } }
2997 { \@preamerr 5 }
2998 }
2999 \exp_args:No \mkpream \g_@@_preamble_tl
3000 \@addtopreamble \empty
3001 \endgroup
3002 \UseTaggingSocket { tbl / colspan } { #1 }
```

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

```
3003 \int_compare:nNnT { #1 } > { \c_one_int }
3004 {
3005 \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
3006 { \int_use:N \c_iRow - \int_eval:n { \c_jCol + 1 } }
3007 \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
```



```

3008 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
3009 {
3010   {
3011     \int_if_zero:nTF { \c@jCol }
3012       { \int_eval:n { \c@iRow + 1 } }
3013       { \int_use:N \c@iRow }
3014   }
3015   { \int_eval:n { \c@jCol + 1 } }
3016   {
3017     \int_if_zero:nTF { \c@jCol }
3018       { \int_eval:n { \c@iRow + 1 } }
3019       { \int_use:N \c@iRow }
3020   }
3021   { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block.

```

3022   { }
3023 }
3024 }

```

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

```

3025 \RenewDocumentCommand { \cellcolor } { 0 { } m }
3026 {
3027   \tl_gput_right:Ne \g_@@_pre_code_before_tl
3028   {
3029     \@@_rectanglecolor [ ##1 ]
3030     { \exp_not:n { ##2 } }
3031     { \int_use:N \c@iRow - \int_use:N \c@jCol }
3032     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
3033   }
3034   \ignorespaces
3035 }

```

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

```

3036 \def \@sharp { #3 }
3037 \@arstrut
3038 \@preamble
3039 \null

```

We add some lines.

```

3040 \int_gadd:Nn \c@jCol { #1 - 1 }
3041 \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
3042   { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3043 \ignorespaces
3044 }

```

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`.

```

3045 \cs_new_protected:Npn \@@_make_m_preamble:n #1
3046 {
3047   \str_case:nnF { #1 }
3048   {
3049     c { \@@_make_m_preamble_i:n #1 }
3050     l { \@@_make_m_preamble_i:n #1 }
3051     r { \@@_make_m_preamble_i:n #1 }
3052     > { \@@_make_m_preamble_ii:nn #1 }
3053     ! { \@@_make_m_preamble_ii:nn #1 }
3054     @ { \@@_make_m_preamble_iii:nn #1 }
3055     | { \@@_make_m_preamble_iii:nn #1 }
3056     p { \@@_make_m_preamble_iv:nnn t #1 }
3057     m { \@@_make_m_preamble_iv:nnn c #1 }

```

```

3058     b { \@@_make_m_preamble_iv:nnn b #1 }
3059     w { \@@_make_m_preamble_v:nnnn { } #1 }
3060     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
3061     \q_stop { }
3062   }
3063   {
3064     \cs_if_exist:cTF { NC @ find @ #1 }
3065     {
3066       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
3067       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
3068     }
3069     {
3070       \str_if_eq:nnTF { #1 } { S }
3071       { \@@_fatal:n { unknown~column~type~S~multicolumn } }
3072       { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
3073     }
3074   }
3075 }

```

For c, l and r

```

3076 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
3077 {
3078   \tl_gput_right:Nn \g_@@_preamble_tl
3079   {
3080     > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
3081     #1
3082     < \@@_cell_end:
3083   }

```

We test for the presence of a <.

```

3084   \@@_make_m_preamble_x:n
3085 }

```

For >, ! and @

```

3086 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
3087 {
3088   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
3089   \@@_make_m_preamble:n
3090 }

```

For |

```

3091 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
3092 {
3093   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
3094   \@@_make_m_preamble:n
3095 }

```

For p, m and b

```

3096 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
3097 {
3098   \tl_gput_right:Nn \g_@@_preamble_tl
3099   {
3100     > {
3101       \@@_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`.

```

3102     \setlength { \l_tmpa_dim } { #3 }
3103     \begin { minipage } [ #1 ] { \l_tmpa_dim }
3104     \mode_leave_vertical:
3105     \arraybackslash
3106     \vrule height \box_ht:N \@@arstrutbox depth \c_zero_dim width \c_zero_dim

```

```

3107     }
3108     c
3109     < {
3110         \vrule height \c_zero_dim depth \box_dp:N \@arstrutbox width \c_zero_dim
3111         \end { minipage }
3112         \@@_cell_end:
3113     }
3114 }

```

We test for the presence of a <.

```

3115 \@@_make_m_preamble_x:n
3116 }

```

For w and W

```

3117 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3118 {
3119     \tl_gput_right:Nn \g_@@_preamble_tl
3120     {
3121         > {
3122             \dim_set:Nn \l_@@_col_width_dim { #4 }
3123             \hbox_set:Nw \l_@@_cell_box
3124             \@@_cell_begin:
3125             \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3126         }
3127         c
3128         < {
3129             \@@_cell_end:
3130             \hbox_set_end:
3131             \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3132             #1
3133             \@@_adjust_size_box:
3134             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3135         }
3136     }

```

We test for the presence of a <.

```

3137 \@@_make_m_preamble_x:n
3138 }

```

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

```

3139 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3140 {
3141     \str_if_eq:nnTF { #1 } { < }
3142     { \@@_make_m_preamble_ix:n }
3143     { \@@_make_m_preamble:n { #1 } }
3144 }
3145 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3146 {
3147     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3148     \@@_make_m_preamble_x:n
3149 }

```

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).

```

3150 \cs_new_protected:Npn \@@_put_box_in_flow:
3151 {
3152     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3153     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3154     \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
3155     { \box_use_drop:N \l_tmpa_box }
3156     { \@@_put_box_in_flow_i: }
3157 }

```

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

```

3158 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3159 {
3160   \pgfpicture
3161     \@@_qpoint:n { row - 1 }
3162     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3163     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3164     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3165     \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).

```

3166     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3167     {
3168       \int_set:Nn \l_tmpa_int
3169       { \str_range:Nnn \l_@@_baseline_tl { 6 } { -1 } }
3170       \bool_lazy_or:nnT
3171       { \int_compare_p:nNn { \l_tmpa_int } < { 1 } }
3172       { \int_compare_p:nNn { \l_tmpa_int } > { \c@iRow + 1 } }
3173       {
3174         \@@_error:n { bad-value-for-baseline-line }
3175         \int_set_eq:NN \l_tmpa_int \c_one_int
3176       }
3177       \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3178     }
3179     {
3180       \str_if_eq:eeTF { \l_@@_baseline_tl } { t }
3181       { \int_set_eq:NN \l_tmpa_int \c_one_int }
3182       {
3183         \str_if_eq:onTF \l_@@_baseline_tl { b }
3184         { \int_set_eq:NN \l_tmpa_int \c@iRow }
3185         { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3186       }
3187       \bool_lazy_or:nnT
3188       { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3189       { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3190       {
3191         \@@_error:n { bad-value-for-baseline }
3192         \int_set_eq:NN \l_tmpa_int \c_one_int
3193       }
3194       \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3195     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3196   }
3197   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3198   \endpgfpicture
3199   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3200   \box_use_drop:N \l_tmpa_box
3201 }

```

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).

```

3202 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3203 {

```

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.

```

3204   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool

```

```

3205 {
3206   \int_compare:nNtT { \c@jCol } > { \c_one_int }
3207   {
3208     \box_set_wd:Nn \l_@@_the_array_box
3209     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3210   }
3211 }

```

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

```

3212 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3213 \bool_if:NT \l_@@_caption_above_bool
3214 {
3215   \tl_if_empty:NF \l_@@_caption_tl
3216   {
3217     \bool_set_false:N \g_@@_caption_finished_bool
3218     \int_gzero:N \c@tabularnote
3219     \@@_insert_caption:

```

If there is one or several commands `\tabularnote` in the caption, we will write in the `aux` file the number of such tabular notes... but only the tabular notes for which the command `\tabularnote` has been used without its optional argument (between square brackets).

```

3220   \int_compare:nNtT { \g_@@_notes_caption_int } > { \c_zero_int }
3221   {
3222     \tl_gput_right:Ne \g_@@_aux_tl
3223     {
3224       \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3225       { \int_use:N \g_@@_notes_caption_int }
3226     }
3227     \int_gzero:N \g_@@_notes_caption_int
3228   }
3229 }
3230 }

```

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

```

3231 \hbox
3232 {
3233   \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.

```

3234   \@@_create_extra_nodes:
3235   \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3236 }

```

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).

```

3237   \bool_lazy_any:nT
3238   {
3239     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3240     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3241     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3242   }
3243   \@@_insert_tabularnotes:
3244   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3245   \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3246   \end { minipage }
3247 }

```

```

3248 \cs_new_protected:Npn \@@_insert_caption:
3249 {
3250   \tl_if_empty:NF \l_@@_caption_tl
3251   {
3252     \cs_if_exist:NTF \@capttype
3253     { \@@_insert_caption_i: }
3254     { \@@_error:n { caption-outside-float } }
3255   }
3256 }

```

```

3257 \cs_new_protected:Npn \@@_insert_caption_i:
3258 {
3259   \group_begin:

```

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

```

3260   \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.

```

3261   \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
3262   \tl_if_empty:NTF \l_@@_short_caption_tl
3263   { \caption }
3264   { \caption [ \l_@@_short_caption_tl ] }
3265   { \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.

```

3266   \bool_if:NF \g_@@_caption_finished_bool
3267   {
3268     \bool_gset_true:N \g_@@_caption_finished_bool
3269     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3270     \int_gzero:N \c@tabularnote
3271   }
3272   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3273   \group_end:
3274 }

```

```

3275 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3276 {
3277   \@@_error_or_warning:n { tabularnote-below-the-tabular }
3278   \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3279 }

```

```

3280 \cs_new_protected:Npn \@@_insert_tabularnotes:
3281 {
3282   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3283   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3284   \skip_vertical:N 0.65ex

```

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

```

3285   \group_begin:
3286   \l_@@_notes_code_before_tl
3287   \tl_if_empty:NF \g_@@_tabularnote_tl
3288   {
3289     \g_@@_tabularnote_tl \par
3290     \tl_gclear:N \g_@@_tabularnote_tl
3291   }

```

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.

```

3292 \int_compare:nNtT { \c@tabularnote } > { \c_zero_int }
3293 {
3294   \bool_if:NTF \l_@@_notes_para_bool
3295   {
3296     \begin { tabularnotes* }
3297       \seq_map_inline:Nn \g_@@_notes_seq
3298       { \@@_one_tabularnote:nn ##1 }
3299       \strut
3300     \end { tabularnotes* }

```

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

```

3301 \par
3302 }
3303 {
3304   \tabularnotes
3305   \seq_map_inline:Nn \g_@@_notes_seq
3306   { \@@_one_tabularnote:nn ##1 }
3307   \strut
3308   \endtabularnotes
3309 }
3310 }
3311 \unskip
3312 \group_end:
3313 \bool_if:NT \l_@@_notes_bottomrule_bool
3314 {
3315   \IfPackageLoadedTF { booktabs }
3316   {

```

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

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

```

3317 \skip_vertical:N \aboverulesep
3318 { \CT@arc@ \hrule height \heavyrulewidth }
3319 }
3320 { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3321 }
3322 \l_@@_notes_code_after_tl
3323 \seq_gclear:N \g_@@_notes_seq
3324 \seq_gclear:N \g_@@_notes_in_caption_seq
3325 \int_gzero:N \c@tabularnote
3326 }

```

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

```

3327 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3328 {
3329   \tl_if_novalue:NTF { #1 }
3330   { \item }
3331   { \item [ \@@_notes_label_in_list:n { #1 } ] }
3332 }

```

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`.

```

3333 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3334 {
3335   \pgfpicture
3336   \@@_qpoint:n { row - 1 }
3337   \dim_gset_eq:NN \g_tmpa_dim \pgf@y

```

```

3338 \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3339 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3340 \endpgfpicture
3341 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3342 \int_if_zero:nT { \l_@@_first_row_int }
3343 {
3344 \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3345 \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3346 }
3347 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3348 }

```

Now, the general case.

```

3349 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3350 {

```

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

```

3351 \str_if_eq:eeT { \l_@@_baseline_tl } { t }
3352 { \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 .

```

3353 \pgfpicture
3354 \@@_qpoint:n { row - 1 }
3355 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3356 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3357 {
3358 \int_set:Nn \l_tmpa_int
3359 {
3360 \str_range:Nnn
3361 \l_@@_baseline_tl
3362 { 6 }
3363 { \tl_count:o \l_@@_baseline_tl }
3364 }
3365 \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3366 }
3367 {
3368 \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3369 \bool_lazy_or:nnT
3370 { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3371 { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3372 {
3373 \@@_error:n { bad~value~for~baseline }
3374 \int_set:Nn \l_tmpa_int 1
3375 }
3376 \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3377 }
3378 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3379 \endpgfpicture
3380 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3381 \int_if_zero:nT { \l_@@_first_row_int }
3382 {
3383 \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3384 \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3385 }
3386 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3387 }

```

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.

```

3388 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3389 {

```


We will compute the real width of both delimiters used.

```

3390 \dim_zero_new:N \l_@@_real_left_delim_dim
3391 \dim_zero_new:N \l_@@_real_right_delim_dim
3392 \hbox_set:Nn \l_tmpb_box
3393 {
3394   \m@th
3395   $ % $
3396   \left #1
3397   \vcenter
3398   {
3399     \vbox_to_ht:nn
3400     { \box_ht_plus_dp:N \l_tmpa_box }
3401     { }
3402   }
3403   \right .
3404   $ % $
3405 }
3406 \dim_set:Nn \l_@@_real_left_delim_dim
3407 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3408 \hbox_set:Nn \l_tmpb_box
3409 {
3410   \m@th
3411   $ % $
3412   \left .
3413   \vbox_to_ht:nn
3414   { \box_ht_plus_dp:N \l_tmpa_box }
3415   { }
3416   \right #2
3417   $ % $
3418 }
3419 \dim_set:Nn \l_@@_real_right_delim_dim
3420 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3421 \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3422 \@@_put_box_in_flow:
3423 \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3424 }

```

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).

```

3425 \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).

```

3426 {
3427   \peek_remove_spaces:n
3428   {
3429     \peek_meaning:NTF \end
3430     { \@@_analyze_end:Nn }
3431     {
3432       \@@_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`).

```

3433 \@@_array:o \g_@@_array_preamble_tl
3434 }
3435 }
3436 }

```

```

3437 {
3438   \@@_create_col_nodes:
3439   \endarray
3440 }

```

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

```

3441 \NewDocumentEnvironment { @@-light-syntax } { b }
3442 {

```

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`.

```

3443   \tl_if_empty:nT { #1 }
3444   { \@@_fatal:n { empty-environment } }
3445   \tl_if_in:nnT { #1 } { & }
3446   { \@@_fatal:n { ampersand-in-light-syntax } }
3447   \tl_if_in:nnT { #1 } { \ }
3448   { \@@_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`.

```

3449   \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3450 }

```

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.

```

3451 {
3452   \@@_create_col_nodes:
3453   \endarray
3454 }
3455 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3456 {
3457   \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).

```

3458   \seq_clear_new:N \l_@@_rows_seq

```

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

```

3459   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3460   \bool_if:NTF \l_@@_light_syntax_expanded_bool
3461   { \seq_set_split:Nee }
3462   { \seq_set_split:Non }
3463   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3464   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3465   \tl_if_empty:NF \l_tmpa_tl
3466   { \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.

```

3467   \int_compare:nNt { \l_@@_last_row_int } = { -1 }
3468   { \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`).

```
3469 \tl_build_begin:N \l_@@_new_body_tl
3470 \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3471 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3472 \@@_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).

```
3473 \seq_map_inline:Nn \l_@@_rows_seq
3474 {
3475   \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3476   \@@_line_with_light_syntax:n { #1 }
3477 }
3478 \tl_build_end:N \l_@@_new_body_tl
3479 \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
3480 {
3481   \int_set:Nn \l_@@_last_col_int
3482     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3483 }
```

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.

```
3484 \@@_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`).

```
3485 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3486 }
3487 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3488 {
3489   \seq_clear_new:N \l_@@_cells_seq
3490   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3491   \int_set:Nn \l_@@_nb_cols_int
3492     {
3493       \int_max:nn
3494         { \l_@@_nb_cols_int }
3495         { \seq_count:N \l_@@_cells_seq }
3496     }
3497   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3498   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3499   \seq_map_inline:Nn \l_@@_cells_seq
3500     { \tl_build_put_right:Nn \l_@@_new_body_tl { & #1 } }
3501 }
3502 \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`.

```
3503 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3504 {
3505   \str_if_eq:eeT { \g_@@_name_env_str } { #2 }
3506     { \@@_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.

```
3507 \end { #2 }
3508 }
```

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`).

```

3509 \cs_new:Npn \@@_create_col_nodes:
3510 {
3511   \crrc
3512   \int_if_zero:nT { \l_@@_first_col_int }
3513   {
3514     \omit
3515     \hbox_overlap_left:n
3516     {
3517       \bool_if:NT \l_@@_code_before_bool
3518       { \pgfsys@markposition { \@@_env: - col - 0 } }
3519       \pgfpicture
3520       \pgfrememberpicturepositiononpagetrue
3521       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3522       \str_if_empty:NF \l_@@_name_str
3523       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3524       \endpgfpicture
3525       \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3526     }
3527     &
3528   }
3529   \omit

```

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

```

3530   \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`).

```

3531   \int_if_zero:nTF { \l_@@_first_col_int }
3532   {
3533     \@@_mark_position:n { 1 }
3534     \pgfpicture
3535     \pgfrememberpicturepositiononpagetrue
3536     \pgfcoordinate { \@@_env: - col - 1 }
3537     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3538     \str_if_empty:NF \l_@@_name_str
3539     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3540     \endpgfpicture
3541   }
3542   {
3543     \bool_if:NT \l_@@_code_before_bool
3544     {
3545       \hbox
3546       {
3547         \skip_horizontal:n { 0.5 \arrayrulewidth }
3548         \pgfsys@markposition { \@@_env: - col - 1 }
3549         \skip_horizontal:n { -0.5 \arrayrulewidth }
3550       }
3551     }
3552     \pgfpicture
3553     \pgfrememberpicturepositiononpagetrue
3554     \pgfcoordinate { \@@_env: - col - 1 }
3555     { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3556     \@@_node_alias:n { 1 }
3557     \endpgfpicture
3558   }

```

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.

```

3559 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3560 \bool_if:NF \l_@@_auto_columns_width_bool
3561 { \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim } }
3562 {
3563   \bool_lazy_and:nnTF
3564     { \l_@@_auto_columns_width_bool }
3565     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3566     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3567     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3568     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3569 }
3570 \skip_horizontal:N \g_tmpa_skip
3571 \hbox
3572 {
3573   \@@_mark_position:n { 2 }
3574   \pgfpicture
3575   \pgfrememberpicturepositiononpagetrue
3576   \pgfcoordinate { \@@_env: - col - 2 }
3577     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3578   \@@_node_alias:n { 2 }
3579   \endpgfpicture
3580 }

```

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.

```

3581 \int_gset_eq:NN \g_tmpa_int \c_one_int
3582 \bool_if:NTF \g_@@_last_col_found_bool
3583 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3584 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3585 {
3586   &
3587   \omit
3588   \int_gincr:N \g_tmpa_int

```

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

```

3589 \skip_horizontal:N \g_tmpa_skip
3590 \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

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

```

3591 \pgfpicture
3592 \pgfrememberpicturepositiononpagetrue
3593 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3594 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3595 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3596 \endpgfpicture
3597 }

```

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).

```

3598 \bool_lazy_or:nnF
3599 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3600 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3601 {
3602   &
3603   \omit
3604   \skip_horizontal:N \g_tmpa_skip
3605   \int_gincr:N \g_tmpa_int
3606   \bool_lazy_any:nF
3607   {
3608     \g_@@_delims_bool
3609     \l_@@_tabular_bool
3610     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3611     \l_@@_exterior_arraycolsep_bool

```

```

3612         \l_@@_bar_at_end_of_pream_bool
3613     }
3614     { \skip_horizontal:n { - \col@sep } }
3615 \bool_if:NT \l_@@_code_before_bool
3616 {
3617     \hbox
3618     {
3619         \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.

```

3620         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3621         { \skip_horizontal:n { - \arraycolsep } }
3622         \pgfsys@markposition
3623         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3624         \skip_horizontal:n { 0.5 \arrayrulewidth }
3625         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3626         { \skip_horizontal:N \arraycolsep }
3627     }
3628 }
3629 \pgfpicture
3630 \pgfrememberpicturepositiononpagetrue
3631 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3632 {
3633     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3634     {
3635         \pgfpoint
3636         { - 0.5 \arrayrulewidth - \arraycolsep }
3637         \c_zero_dim
3638     }
3639     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3640 }
3641 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3642 \endpgfpicture
3643 }

3644 \bool_if:NT \g_@@_last_col_found_bool
3645 {
3646     \hbox_overlap_right:n
3647     {
3648         \skip_horizontal:N \g_@@_width_last_col_dim
3649         \skip_horizontal:N \col@sep
3650         \bool_if:NT \l_@@_code_before_bool
3651         {
3652             \pgfsys@markposition
3653             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3654         }
3655         \pgfpicture
3656         \pgfrememberpicturepositiononpagetrue
3657         \pgfcoordinate
3658         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3659         \pgfpointorigin
3660         \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3661         \endpgfpicture
3662     }
3663 }
3664 }

3665 \cs_new_protected:Npn \@@_mark_position:n #1
3666 {
3667     \bool_if:NT \l_@@_code_before_bool

```

```

3668     {
3669         \hbox
3670         {
3671             \skip_horizontal:n { -0.5 \arrayrulewidth }
3672             \pgfsys@markposition { \@@_env: - col - #1 }
3673             \skip_horizontal:n { 0.5 \arrayrulewidth }
3674         }
3675     }
3676 }
3677 \cs_new_protected:Npn \@@_node_alias:n #1
3678 {
3679     \str_if_empty:NF \l_@@_name_str
3680     { \pgfnodelias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3681 }

```

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

```

3682 \tl_const:Nn \c_@@_preamble_first_col_tl
3683 {
3684     >
3685     {

```

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

```

3686         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3687         \bool_gset_true:N \g_@@_after_col_zero_bool
3688         \@@_begin_of_row:
3689         \hbox_set:Nw \l_@@_cell_box
3690         \@@_math_toggle:
3691         \@@_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”.

```

3692         \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3693         {
3694             \bool_lazy_or:nnT
3695             { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3696             { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3697             {
3698                 \l_@@_code_for_first_col_tl
3699                 \xglobal \colorlet { nicematrix-first-col } { . }
3700             }
3701         }
3702     }

```

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`.

```

3703     l
3704     <
3705     {
3706         \@@_math_toggle:
3707         \hbox_set_end:
3708         \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3709         \@@_adjust_size_box:
3710         \@@_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.

```

3711         \dim_gset:Nn \g_@@_width_first_col_dim
3712         { \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.

```

3713         \hbox_overlap_left:n
3714         {

```

```

3715         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3716         { \@@_node_cell: }
3717         { \box_use_drop:N \l_@@_cell_box }
3718         \skip_horizontal:N \l_@@_left_delim_dim
3719         \skip_horizontal:N \l_@@_left_margin_dim
3720         \skip_horizontal:N \l_@@_extra_left_margin_dim
3721     }
3722     \bool_gset_false:N \g_@@_empty_cell_bool
3723     \skip_horizontal:n { -2 \col@sep }
3724 }
3725 }

```

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

```

3726 \tl_const:Nn \c_@@_preamble_last_col_tl
3727 {
3728     >
3729     {
3730         \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).

```

3731         \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.

```

3732         \bool_gset_true:N \g_@@_last_col_found_bool
3733         \int_gincr:N \c@jCol
3734         \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3735         \hbox_set:Nw \l_@@_cell_box
3736         \@@_math_toggle:
3737         \@@_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”.

```

3738         \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3739         {
3740             \bool_lazy_or:nnT
3741             { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3742             { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3743             {
3744                 \l_@@_code_for_last_col_tl
3745                 \xglobal \colorlet { nicematrix-last-col } { . }
3746             }
3747         }
3748     }
3749     1
3750     <
3751     {
3752         \@@_math_toggle:
3753         \hbox_set_end:
3754         \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3755         \@@_adjust_size_box:
3756         \@@_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.

```

3757         \dim_gset:Nn \g_@@_width_last_col_dim
3758         { \dim_max:nn { \g_@@_width_last_col_dim } { \box_wd:N \l_@@_cell_box } }
3759         \skip_horizontal:n { -2 \col@sep }

```

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

```

3760         \hbox_overlap_right:n
3761         {
3762             \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3763             {
3764                 \skip_horizontal:N \l_@@_right_delim_dim
3765                 \skip_horizontal:N \l_@@_right_margin_dim

```



```

3766         \skip_horizontal:N \l_@@_extra_right_margin_dim
3767         \@@_node_cell:
3768     }
3769 }
3770 \bool_gset_false:N \g_@@_empty_cell_bool
3771 }
3772 }

```

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

```

3773 \NewDocumentEnvironment { NiceArray } { }
3774 {
3775     \bool_gset_false:N \g_@@_delims_bool
3776     \str_if_empty:NT \g_@@_name_env_str
3777     { \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).

```

3778     \NiceArrayWithDelims . .
3779 }
3780 { \endNiceArrayWithDelims }

```

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

```

3781 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3782 {
3783     \NewDocumentEnvironment { #1 NiceArray } { }
3784     {
3785         \bool_gset_true:N \g_@@_delims_bool
3786         \str_if_empty:NT \g_@@_name_env_str
3787         { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3788         \@@_test_if_math_mode:
3789         \NiceArrayWithDelims #2 #3
3790     }
3791     { \endNiceArrayWithDelims }
3792 }
3793 \@@_def_env:NNN p ( )
3794 \@@_def_env:NNN b [ ]
3795 \@@_def_env:NNN B \{ \}
3796 \@@_def_env:NNN v \vert \vert
3797 \@@_def_env:NNN V \Vert \Vert

```

13 The environment `{NiceMatrix}` and its variants

```

3798 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3799 {
3800     \bool_set_false:N \l_@@_preamble_bool
3801     \tl_clear:N \l_tmpa_tl
3802     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3803     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3804     \tl_put_right:Nn \l_tmpa_tl
3805     {
3806         *
3807         {
3808             \int_case:nnF \l_@@_last_col_int
3809             {
3810                 { -2 } { \c@MaxMatrixCols }
3811                 { -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).

```

3812     }
3813     { \int_eval:n { \l_@@_last_col_int - 1 } }
3814   }
3815   { #2 }
3816 }
3817 \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3818 \exp_args:No \l_tmpb_tl \l_tpa_tl
3819 }
3820 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3821 \clist_map_inline:nn { p , b , B , v , V }
3822 {
3823   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3824   {
3825     \bool_gset_true:N \g_@@_delims_bool
3826     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3827     \int_if_zero:nT { \l_@@_last_col_int }
3828     {
3829       \bool_set_true:N \l_@@_last_col_without_value_bool
3830       \int_set:Nn \l_@@_last_col_int { -1 }
3831     }
3832     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3833     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3834   }
3835   { \use:c { end #1 NiceArray } }
3836 }

```

We define also an environment {NiceMatrix}

```

3837 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3838 {
3839   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3840   \int_if_zero:nT { \l_@@_last_col_int }
3841   {
3842     \bool_set_true:N \l_@@_last_col_without_value_bool
3843     \int_set:Nn \l_@@_last_col_int { -1 }
3844   }
3845   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3846   \bool_lazy_or:nnT
3847   { \clist_if_empty_p:N \l_@@_vlines_clist }
3848   { \l_@@_except_borders_bool }
3849   { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3850   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3851 }
3852 { \endNiceArray }

```

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

```

3853 \cs_new_protected:Npn \@@_NotEmpty:
3854 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3855 \NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
3856 {

```

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.

```

3857   \dim_compare:nNnT { \l_@@_width_dim } = { \c_zero_dim }
3858   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3859   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3860   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3861   \tl_if_empty:NF \l_@@_short_caption_tl

```

```

3862 {
3863   \tl_if_empty:NT \l_@@_caption_tl
3864   {
3865     \@@_error_or_warning:n { short-caption-without~caption }
3866     \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3867   }
3868 }
3869 \tl_if_empty:NF \l_@@_label_tl
3870 {
3871   \tl_if_empty:NT \l_@@_caption_tl
3872   { \@@_error_or_warning:n { label~without~caption } }
3873 }
3874 \NewDocumentEnvironment { TabularNote } { b }
3875 {
3876   \bool_if:NTF \l_@@_in_code_after_bool
3877   { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3878   {
3879     \tl_if_empty:NF \g_@@_tabularnote_tl
3880     { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3881     \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3882   }
3883 }
3884 { }
3885 \@@_settings_for_tabular:
3886 \NiceArray { #2 }
3887 }
3888 { \endNiceArray }
3889 \cs_new_protected:Npn \@@_settings_for_tabular:
3890 {
3891   \bool_set_true:N \l_@@_tabular_bool
3892   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3893   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3894   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3895 }

3896 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3897 {
3898   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3899   \dim_set:Nn \l_@@_width_dim { #1 }
3900   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3901   \@@_settings_for_tabular:
3902   \NiceArray { #3 }
3903 }
3904 {
3905   \endNiceArray
3906   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3907   { \@@_error:n { NiceTabularX~without~X } }
3908 }

3909 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3910 {
3911   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3912   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3913   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3914   \@@_settings_for_tabular:
3915   \NiceArray { #3 }
3916 }
3917 { \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`).

```

3918 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3919 {
3920   \bool_lazy_all:nT
3921   {
3922     { \dim_compare_p:nNn { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim } }
3923     { \l_@@_hvlines_bool }
3924     { ! \g_@@_delims_bool }
3925     { ! \l_@@_except_borders_bool }
3926   }
3927   {
3928     \bool_set_true:N \l_@@_except_borders_bool
3929     \clist_if_empty:NF \l_@@_corners_clist
3930     { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3931     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3932     {
3933       \@@_stroke_block:nnn
3934       {
3935         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3936         draw = \l_@@_rules_color_tl
3937       }
3938       { 1-1 }
3939       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3940     }
3941   }
3942 }

3943 \cs_new_protected:Npn \@@_after_array:
3944 {

```

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`.

```

3945   \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3946   \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.

```

3947   \bool_if:NT \g_@@_last_col_found_bool
3948   { \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`.

```

3949   \bool_if:NT \l_@@_last_col_without_value_bool
3950   { \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.

```

3951   \bool_if:NT \l_@@_last_row_without_value_bool
3952   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3953 \tl_gput_right:Ne \g_@@_aux_tl
3954 {
3955   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3956   {
3957     \int_use:N \l_@@_first_row_int ,
3958     \int_use:N \c@iRow ,
3959     \int_use:N \g_@@_row_total_int ,
3960     \int_use:N \l_@@_first_col_int ,
3961     \int_use:N \c@jCol ,
3962     \int_use:N \g_@@_col_total_int
3963   }
3964 }
3965 \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`).

```

3966 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3967 {
3968   \tl_gput_right:Ne \g_@@_aux_tl
3969   {
3970     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3971     { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }
3972   }
3973 }
3974 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3975 {
3976   \tl_gput_right:Ne \g_@@_aux_tl
3977   {
3978     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3979     { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
3980     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3981     { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
3982   }
3983 }

```

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

```

3984 \@@_create_diag_nodes:

```

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

```

3985 \pgfpicture
3986 \@@_create_aliases_last:
3987 \str_if_empty:NF \l_@@_name_str { \@@_create_alias_nodes: }
3988 \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.

```

3989 \bool_if:NT \l_@@_parallelize_diags_bool
3990 {
3991   \int_gzero:N \g_@@_ddots_int
3992   \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.

```

3993 \dim_gzero:N \g_@@_delta_x_one_dim
3994 \dim_gzero:N \g_@@_delta_y_one_dim
3995 \dim_gzero:N \g_@@_delta_x_two_dim

```

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

```

3996      \dim_gzero:N \g_@@_delta_y_two_dim
3997    }
3998    \bool_set_false:N \l_@@_initial_open_bool
3999    \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.

```

4000    \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

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

```

4001    \@@_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_clist` which will contain all the cells which are empty (and not in a block) considered in the corners of the array.

```

4002    \clist_if_empty:NF \l_@@_corners_clist
4003    {
4004      \bool_if:NTF \l_@@_no_cell_nodes_bool
4005      { \@@_error:n { corners~with~no~cell~nodes } }
4006      { \@@_compute_corners: }
4007    }

```

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

```

4008    \seq_gconcat:NNN \g_@@_pos_of_blocks_seq
4009    \g_@@_pos_of_blocks_seq
4010    \g_@@_future_pos_of_blocks_seq
4011    \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-*}`).

```

4012    \@@_adjust_pos_of_blocks_seq:
4013    \@@_deal_with_rounded_corners:
4014    \clist_if_empty:NF \l_@@_hlines_clist { \@@_draw_hlines: }
4015    \clist_if_empty:NF \l_@@_vlines_clist { \@@_draw_vlines: }

```

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

```

4016    \IfPackageLoadedT { tikz }
4017    {
4018      \tikzset
4019      {
4020        every-picture / .style =
4021        {
4022          overlay ,
4023          remember~picture ,
4024          name~prefix = \@@_env: -
4025        }
4026      }
4027    }
4028    \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
4029    \cs_set_eq:NN \SubMatrix \@@_SubMatrix
4030    \cs_set_eq:NN \UnderBrace \@@_UnderBrace
4031    \cs_set_eq:NN \OverBrace \@@_OverBrace
4032    \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
4033    \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
4034    \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.

```
4035 \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
4036 \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.

```
4037 \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).

```
4038 \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`).

```
4039 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
4040 { \@@_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:`.

```
4041 \bool_set_true:N \l_@@_in_code_after_bool
4042 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4043 \scan_stop:
4044 \tl_gclear:N \g_nicematrix_code_after_tl
4045 \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.

```
4046 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
4047 \tl_if_empty:NF \g_@@_pre_code_before_tl
4048 {
4049   \tl_gput_right:Ne \g_@@_aux_tl
4050   {
4051     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
4052     { \exp_not:o \g_@@_pre_code_before_tl }
4053   }
4054   \tl_gclear:N \g_@@_pre_code_before_tl
4055 }
4056 \tl_if_empty:NF \g_nicematrix_code_before_tl
4057 {
4058   \tl_gput_right:Ne \g_@@_aux_tl
4059   {
4060     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
4061     { \exp_not:o \g_nicematrix_code_before_tl }
4062   }
4063   \tl_gclear:N \g_nicematrix_code_before_tl
4064 }

4065 \str_gclear:N \g_@@_name_env_str
4066 \@@_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`.

```
4067 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
4068 }
```

¹³e.g. `\color[rgb]{0.5,0.5,0}`

```

4069 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
4070 {
4071   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
4072   \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }

```

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

```

4073   \dim_set:Nn \l_@@_xdots_shorten_start_dim
4074     { 0.6 \l_@@_xdots_shorten_start_dim }
4075   \dim_set:Nn \l_@@_xdots_shorten_end_dim
4076     { 0.6 \l_@@_xdots_shorten_end_dim }
4077 }

```

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`.

```

4078 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4079 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

```

```

4080 \cs_new_protected:Npn \@@_create_alias_nodes:
4081 {
4082   \int_step_inline:nn { \c@iRow }
4083   {
4084     \pgfnodealias
4085       { \l_@@_name_str - ##1 - last }
4086       { \@@_env: - ##1 - \int_use:N \c@jCol }
4087   }
4088   \int_step_inline:nn { \c@jCol }
4089   {
4090     \pgfnodealias
4091       { \l_@@_name_str - last - ##1 }
4092       { \@@_env: - \int_use:N \c@iRow - ##1 }
4093   }
4094   \pgfnodealias
4095     { \l_@@_name_str - last - last }
4096     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
4097 }

```

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).

```

4098 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4099 {
4100   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4101     { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4102 }

```

The following command must *not* be protected.

```

4103 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4104 {
4105   { #1 }
4106   { #2 }
4107   {
4108     \int_compare:nNnTF { #3 } > { 98 }
4109     { \int_use:N \c@iRow }
4110     { #3 }
4111   }

```



```

4112 {
4113   \int_compare:nNnTF { #4 } > { 98 }
4114     { \int_use:N \c@jCol }
4115     { #4 }
4116 }
4117 { #5 }
4118 }

```

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).

```

4119 \hook_gput_code:nnn { begindocument } { . }
4120 {
4121   \cs_new_protected:Npe \@@_draw_dotted_lines:
4122     {
4123       \c_@@_pgfortikzpicture_tl
4124       \@@_draw_dotted_lines_i:
4125       \c_@@_endpgfortikzpicture_tl
4126     }
4127 }

```

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

```

4128 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4129 {
4130   \pgfrememberpicturepositiononpagetrue
4131   \pgf@relevantforpicturesizefalse
4132   \g_@@_HVDotsfor_lines_tl
4133   \g_@@_VDots_lines_tl
4134   \g_@@_Ddots_lines_tl
4135   \g_@@_Iddots_lines_tl
4136   \g_@@_Cdots_lines_tl
4137   \g_@@_Ldots_lines_tl
4138 }

4139 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4140 {
4141   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4142   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4143 }

```

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

```

4144 \pgfdeclareshape { @@_diag_node }
4145 {
4146   \savedanchor { \five }
4147   {
4148     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4149     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4150   }
4151   \anchor { 5 } { \five }
4152   \anchor { center } { \pgfpointorigin }
4153   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4154   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4155   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4156   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4157   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4158   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4159   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4160   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4161   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4162   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4163 }

```

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).

```

4164 \cs_new_protected:Npn \@@_create_diag_nodes:
4165 {
4166   \pgfpicture
4167   \pgfrememberpicturepositiononpagetrue
4168   \int_step_inline:nn { \int_max:nn { \c@iRow } { \c@jCol } }
4169   {
4170     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4171     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4172     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4173     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4174     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4175     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4176     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4177     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4178     \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.

```

4179     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4180     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4181     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4182     \str_if_empty:NF \l_@@_name_str
4183     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4184   }

```

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

```

4185   \int_set:Nn \l_tmpa_int { \int_max:nn { \c@iRow } { \c@jCol } + 1 }
4186   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4187   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4188   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4189   \pgfcoordinate
4190   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4191   \pgfnodealias
4192   { \@@_env: - last }
4193   { \@@_env: - \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
4194   \str_if_empty:NF \l_@@_name_str
4195   {
4196     \pgfnodealias
4197     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4198     { \@@_env: - \int_use:N \l_tmpa_int }
4199     \pgfnodealias
4200     { \l_@@_name_str - last }
4201     { \@@_env: - last }
4202   }
4203   \endpgfpicture
4204 }

```

16 We draw the dotted lines

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

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

The command `\l_@@_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 \l_@@_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.

```

4205 \cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
4206 {

```

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

```

4207   \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4208   \l_@@_initial_i_int = #1
4209   \l_@@_initial_j_int = #2
4210   \l_@@_final_i_int = #1
4211   \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.

```

4212     \let \l_@@_stop_loop_bool \c_false_bool
4213     \bool_do_until:Nn \l_@@_stop_loop_bool
4214     {

```

We test if we are still in the matrix.

```

4215         \advance \l_@@_final_i_int by #3
4216         \advance \l_@@_final_j_int by #4
4217         \let \l_@@_final_open_bool \c_false_bool
4218         \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4219             \if_int_compare:w #3 = \c_one_int
4220                 \let \l_@@_final_open_bool \c_true_bool
4221             \else:
4222                 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4223                     \let \l_@@_final_open_bool \c_true_bool
4224                 \fi:
4225             \fi:
4226         \else:
4227             \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4228                 \if_int_compare:w #4 = -1
4229                     \let \l_@@_final_open_bool \c_true_bool
4230                 \fi:
4231             \else:
4232                 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4233                     \if_int_compare:w #4 = \c_one_int
4234                         \let \l_@@_final_open_bool \c_true_bool
4235                     \fi:
4236                 \fi:
4237             \fi:
4238         \fi:
4239         \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.

```

4240     {

```

We do a step backwards.

```

4241         \advance \l_@@_final_i_int by - #3
4242         \advance \l_@@_final_j_int by - #4
4243         \let \l_@@_stop_loop_bool \c_true_bool
4244     }

```

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`.

```

4245     {
4246         \cs_if_exist:cTF
4247         {
4248             @@ _ dotted _
4249             \int_use:N \l_@@_final_i_int -
4250             \int_use:N \l_@@_final_j_int
4251         }
4252         {
4253             \advance \l_@@_final_i_int by - #3
4254             \advance \l_@@_final_j_int by - #4
4255             \let \l_@@_final_open_bool \c_true_bool
4256             \let \l_@@_stop_loop_bool \c_true_bool
4257         }
4258         {
4259             \cs_if_exist:cTF
4260             {
4261                 pgf @ sh @ ns @ \@@_env:
4262                 - \int_use:N \l_@@_final_i_int
4263                 - \int_use:N \l_@@_final_j_int
4264             }

```

```

4265         { \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.

```

4266     {
4267         \cs_set_nopar:cpn
4268         {
4269             @@ _ dotted _
4270             \int_use:N \l_@@_final_i_int -
4271             \int_use:N \l_@@_final_j_int
4272         }
4273     { }
4274 }
4275 }
4276 }
4277 }

```

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

```

4278     \let \l_@@_stop_loop_bool \c_false_bool

```

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

```

4279     \l_tmpa_int = \l_@@_col_min_int
4280     \advance \l_tmpa_int by -1
4281     \bool_do_until:Nn \l_@@_stop_loop_bool
4282     {
4283         \advance \l_@@_initial_i_int by - #3
4284         \advance \l_@@_initial_j_int by - #4
4285         \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.

```

4286         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4287         \if_int_compare:w #3 = \c_one_int
4288             \let \l_@@_initial_open_bool \c_true_bool
4289         \else:

```

`\l_tmpa_int` contains `\l_@@_col_min_int - 1` (only for efficiency).

```

4290             \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4291                 \let \l_@@_initial_open_bool \c_true_bool
4292             \fi:
4293         \fi:
4294     \else:
4295         \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4296         \if_int_compare:w #4 = \c_one_int
4297             \let \l_@@_initial_open_bool \c_true_bool
4298         \fi:
4299     \else:
4300         \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4301         \if_int_compare:w #4 = -1
4302             \let \l_@@_initial_open_bool \c_true_bool
4303         \fi:
4304     \fi:
4305     \fi:
4306 \fi:
4307 \bool_if:NTF \l_@@_initial_open_bool
4308 {
4309     \advance \l_@@_initial_i_int by #3
4310     \advance \l_@@_initial_j_int by #4

```

```

4311     \let \l_@@_stop_loop_bool \c_true_bool
4312   }
4313   {
4314     \cs_if_exist:cTF
4315     {
4316       @@ _ dotted _
4317       \int_use:N \l_@@_initial_i_int -
4318       \int_use:N \l_@@_initial_j_int
4319     }
4320     {
4321       \advance \l_@@_initial_i_int by #3
4322       \advance \l_@@_initial_j_int by #4
4323       \let \l_@@_initial_open_bool \c_true_bool
4324       \let \l_@@_stop_loop_bool \c_true_bool
4325     }
4326     {
4327       \cs_if_exist:cTF
4328       {
4329         pgf @ sh @ ns @ \@@_env:
4330         - \int_use:N \l_@@_initial_i_int
4331         - \int_use:N \l_@@_initial_j_int
4332       }
4333       { \let \l_@@_stop_loop_bool \c_true_bool }
4334       {
4335         \cs_set_nopar:cpn
4336         {
4337           @@ _ dotted _
4338           \int_use:N \l_@@_initial_i_int -
4339           \int_use:N \l_@@_initial_j_int
4340         }
4341         { }
4342       }
4343     }
4344   }
4345 }

```

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.

```

4346   \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4347   {
4348     { \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`.

```

4349     { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4350     { \int_use:N \l_@@_final_i_int }
4351     { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4352     { }
4353   }
4354 }

```

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.

```

4355   \cs_new_protected:Npn \@@_open_shorten:
4356   {
4357     \bool_if:NT \l_@@_initial_open_bool
4358     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4359     \bool_if:NT \l_@@_final_open_bool
4360     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4361   }

```


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).

```

4362 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4363 {
4364   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4365   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4366   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4367   \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`.

```

4368   \seq_if_empty:NF \g_@@_submatrix_seq
4369   {
4370     \seq_map_inline:Nn \g_@@_submatrix_seq
4371       { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4372   }
4373 }

```

#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.

```

4374 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4375 {
4376   \if_int_compare:w #3 > #1
4377   \else:
4378     \if_int_compare:w #1 > #5
4379     \else:
4380       \if_int_compare:w #4 > #2
4381       \else:
4382         \if_int_compare:w #2 > #6
4383         \else:
4384           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4385           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4386           \if_int_compare:w \l_@@_row_max_int > #5 \l_@@_row_max_int = #5 \fi:
4387           \if_int_compare:w \l_@@_col_max_int > #6 \l_@@_col_max_int = #6 \fi:
4388         \fi:
4389       \fi:
4390     \fi:
4391   \fi:
4392 }

```

```

4393 \cs_new_protected:Npn \@@_set_initial_coords:
4394 {
4395   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4396   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4397 }
4398 \cs_new_protected:Npn \@@_set_final_coords:
4399 {
4400   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4401   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4402 }
4403 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4404 {
4405   \pgfpointanchor
4406   {
4407     \@@_env:
4408     - \int_use:N \l_@@_initial_i_int
4409     - \int_use:N \l_@@_initial_j_int
4410   }
4411   { #1 }
4412   \@@_set_initial_coords:
4413 }
4414 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4415 {
4416   \pgfpointanchor
4417   {
4418     \@@_env:
4419     - \int_use:N \l_@@_final_i_int
4420     - \int_use:N \l_@@_final_j_int
4421   }
4422   { #1 }
4423   \@@_set_final_coords:
4424 }
4425 \cs_new_protected:Npn \@@_open_x_initial_dim:
4426 {
4427   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4428   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4429   {
4430     \cs_if_exist:cT
4431     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4432     {
4433       \pgfpointanchor
4434       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4435       { west }
4436       \dim_set:Nn \l_@@_x_initial_dim
4437       { \dim_min:nn { \l_@@_x_initial_dim } { \pgf@x } }
4438     }
4439   }

```

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

```

4440   \dim_compare:nNnT { \l_@@_x_initial_dim } = { \c_max_dim }
4441   {
4442     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4443     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4444     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4445   }
4446 }
4447 \cs_new_protected:Npn \@@_open_x_final_dim:
4448 {
4449   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4450   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4451   {
4452     \cs_if_exist:cT
4453     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }

```

```

4454     {
4455         \pgfpointanchor
4456         { \l_@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4457         { east }
4458         \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
4459         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4460     }
4461 }

```

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

```

4462     \dim_compare:nNnT { \l_@@_x_final_dim } = { - \c_max_dim }
4463     {
4464         \l_@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4465         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4466         \dim_sub:Nn \l_@@_x_final_dim \col@sep
4467     }
4468 }

```

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.

```

4469 \cs_new_protected:Npn \l_@@_draw_Ldots:nnn #1 #2 #3
4470 {
4471     \l_@@_adjust_to_submatrix:nn { #1 } { #2 }
4472     \cs_if_free:cT { @ _ dotted _ #1 - #2 }
4473     {
4474         \l_@@_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.

```

4475     \bool_if:NT \g_@@_aux_found_bool
4476     {
4477         \group_begin:
4478         \l_@@_open_shorten:
4479         \int_if_zero:nTF { #1 }
4480         { \color { nicematrix-first-row } }
4481         {

```

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.

```

4482         \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4483         { \color { nicematrix-last-row } }
4484     }
4485     \keys_set:nn { nicematrix / xdots } { #3 }
4486     \l_@@_color:o \l_@@_xdots_color_tl
4487     \l_@@_actually_draw_Ldots:
4488     \group_end:
4489 }
4490 }
4491 }

```

The command `\l_@@_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`.

```

4492 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4493 {
4494   \bool_if:NTF \l_@@_initial_open_bool
4495   {
4496     \@@_open_x_initial_dim:
4497     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4498     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4499   }
4500   { \@@_set_initial_coords_from_anchor:n { base-east } }
4501   \bool_if:NTF \l_@@_final_open_bool
4502   {
4503     \@@_open_x_final_dim:
4504     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4505     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4506   }
4507   { \@@_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.

```

4508   \bool_lazy_all:nTF
4509   {
4510     \l_@@_initial_open_bool
4511     \l_@@_final_open_bool
4512     { \int_compare_p:nNn { \l_@@_initial_i_int } = { \l_@@_last_row_int } }
4513   }
4514   {
4515     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4516     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4517   }

```

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 (?).

```

4518   {
4519     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4520     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4521   }
4522   \@@_draw_line:
4523 }

```

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.

```

4524 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4525 {
4526   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4527   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4528   {
4529     \@@_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.

```

4530   \bool_if:NT \g_@@_aux_found_bool
4531   {
4532     \group_begin:
4533     \@@_open_shorten:
4534     \int_if_zero:nTF { #1 }
4535     { \color { nicematrix-first-row } }
4536     {

```

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.

```

4537     \int_compare:nNnT { #1 } = { \l_@@_last_row_int }

```

```

4538         { \color { nicematrix-last-row } }
4539     }
4540     \keys_set:nn { nicematrix / xdots } { #3 }
4541     \@@_color:o \l_@@_xdots_color_tl
4542     \@@_actually_draw_Cdots:
4543     \group_end:
4544 }
4545 }
4546 }

```

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.`

```

4547 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4548 {
4549     \bool_if:NTF \l_@@_initial_open_bool
4550     { \@@_open_x_initial_dim: }
4551     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4552     \bool_if:NTF \l_@@_final_open_bool
4553     { \@@_open_x_final_dim: }
4554     { \@@_set_final_coords_from_anchor:n { mid-west } }
4555     \bool_lazy_and:nnTF
4556     { \l_@@_initial_open_bool }
4557     { \l_@@_final_open_bool }
4558     {
4559         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4560         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4561         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4562         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4563         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4564     }
4565     {
4566         \bool_if:NT \l_@@_initial_open_bool
4567         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4568         \bool_if:NT \l_@@_final_open_bool
4569         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4570     }
4571     \@@_draw_line:
4572 }
4573 \cs_new_protected:Npn \@@_open_y_initial_dim:
4574 {
4575     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4576     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4577     {
4578         \cs_if_exist:cT
4579         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4580         {
4581             \pgfpointanchor
4582             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4583             { north }
4584             \dim_compare:nNnT { \pgf@y } > { \l_@@_y_initial_dim }
4585             { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4586         }
4587     }

```

```

4588 \dim_compare:nNnT { \l_@@_y_initial_dim } = { - \c_max_dim }
4589 {
4590   \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4591   \dim_set:Nn \l_@@_y_initial_dim
4592   {
4593     \fp_to_dim:n
4594     {
4595       \pgf@y
4596       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4597     }
4598   }
4599 }
4600 }

4601 \cs_new_protected:Npn \@@_open_y_final_dim:
4602 {
4603   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4604   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4605   {
4606     \cs_if_exist:cT
4607     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4608     {
4609       \pgfpointanchor
4610       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4611       { south }
4612       \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
4613       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4614     }
4615   }
4616   \dim_compare:nNnT { \l_@@_y_final_dim } = { \c_max_dim }
4617   {
4618     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4619     \dim_set:Nn \l_@@_y_final_dim
4620     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4621   }
4622 }

```

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.

```

4623 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4624 {
4625   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4626   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4627   {
4628     \@@_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.

```

4629   \bool_if:NT \g_@@_aux_found_bool
4630   {
4631     \group_begin:
4632     \@@_open_shorten:
4633     \int_if_zero:nTF { #2 }
4634     { \color { nicematrix-first-col } }
4635     {
4636       \int_compare:nNnT { #2 } = { \l_@@_last_col_int }
4637       { \color { nicematrix-last-col } }
4638     }
4639     \keys_set:nn { nicematrix / xdots } { #3 }
4640     \@@_color:o \l_@@_xdots_color_tl
4641     \bool_if:NTF \l_@@_Vbrace_bool
4642     { \@@_actually_draw_Vbrace: }
4643     { \@@_actually_draw_Vdots: }
4644     \group_end:
4645   }

```

```

4646     }
4647 }

```

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`.

```

4648 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4649 {
4650   \bool_lazy_and:nnTF { \l_@@_initial_open_bool } { \l_@@_final_open_bool }
4651   { \@@_actually_draw_Vdots_i: }
4652   { \@@_actually_draw_Vdots_ii: }
4653   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4654   \@@_draw_line:
4655 }

```

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

```

4656 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4657 {
4658   \@@_open_y_initial_dim:
4659   \@@_open_y_final_dim:
4660   \int_if_zero:nTF { \l_@@_initial_j_int }

```

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

```

4661   {
4662     \@@_qpoint:n { col - 1 }
4663     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4664     \dim_sub:Nn \l_@@_x_initial_dim
4665     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4666   }
4667   {
4668     \bool_lazy_and:nnTF
4669     { \int_compare_p:nNn { \l_@@_last_col_int } > { -2 } }
4670     {
4671       \int_compare_p:nNn
4672       { \l_@@_initial_j_int } = { \g_@@_col_total_int }
4673     }

```

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

```

4674   {
4675     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4676     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4677     \dim_add:Nn \l_@@_x_initial_dim
4678     { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4679   }

```

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

```

4680   {
4681     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4682     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4683     \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4684     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4685   }
4686 }
4687 }

```

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.

```

4688 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4689 {
4690   \bool_set_false:N \l_tmpa_bool
4691   \bool_if:NF \l_@@_initial_open_bool
4692   {
4693     \bool_if:NF \l_@@_final_open_bool
4694     {
4695       \@@_set_initial_coords_from_anchor:n { south~west }
4696       \@@_set_final_coords_from_anchor:n { north~west }
4697       \bool_set:Nn \l_tmpa_bool
4698       {
4699         \dim_compare_p:nNn
4700         { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4701       }
4702     }
4703   }

```

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

```

4704   \bool_if:NTF \l_@@_initial_open_bool
4705   {
4706     \@@_open_y_initial_dim:
4707     \@@_set_final_coords_from_anchor:n { north }
4708     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4709   }
4710   {
4711     \@@_set_initial_coords_from_anchor:n { south }
4712     \bool_if:NTF \l_@@_final_open_bool
4713     { \@@_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.

```

4714   {
4715     \@@_set_final_coords_from_anchor:n { north }
4716     \dim_compare:nNnF { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4717     {
4718       \dim_set:Nn \l_@@_x_initial_dim
4719       {
4720         \bool_if:NTF \l_tmpa_bool { \dim_min:nn } { \dim_max:nn }
4721         \l_@@_x_initial_dim \l_@@_x_final_dim
4722       }
4723     }
4724   }
4725 }
4726 }

```

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`.


```

4727 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4728 {
4729   \bool_if:NTF \l_@@_initial_open_bool
4730     { \@@_open_y_initial_dim: }
4731     { \@@_set_initial_coords_from_anchor:n { south } }
4732   \bool_if:NTF \l_@@_final_open_bool
4733     { \@@_open_y_final_dim: }
4734     { \@@_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.

```

4735   \int_if_zero:nTF { \l_@@_initial_j_int }
4736   {
4737     \@@_qpoint:n { col - 1 }
4738     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4739     \dim_sub:Nn \l_@@_x_initial_dim
4740       { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4741   }

```

Elsewhere, the brace must be drawn left flush.

```

4742   {
4743     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4744     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4745     \dim_add:Nn \l_@@_x_initial_dim
4746       { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4747   }

```

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

```

4748   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4749   \@@_draw_line:
4750 }

```

```

4751 \cs_new:Npn \@@_colsep:
4752 { \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.

```

4753 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4754 {
4755   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4756   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4757   {
4758     \@@_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.

```

4759   \bool_if:NT \g_@@_aux_found_bool
4760   {
4761     \group_begin:
4762       \@@_open_shorten:
4763       \keys_set:nn { nicematrix / xdots } { #3 }
4764       \@@_color:o \l_@@_xdots_color_tl
4765       \@@_actually_draw_Ddots:
4766     \group_end:
4767   }
4768 }
4769 }

```

The command `\l_@@_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.`

```

4770 \cs_new_protected:Npn \l_@@_actually_draw_Ddots:
4771 {
4772   \bool_if:NTF \l_@@_initial_open_bool
4773   {
4774     \@@_open_y_initial_dim:
4775     \@@_open_x_initial_dim:
4776   }
4777   { \@@_set_initial_coords_from_anchor:n { south-east } }
4778   \bool_if:NTF \l_@@_final_open_bool
4779   {
4780     \@@_open_x_final_dim:
4781     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4782   }
4783   { \@@_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.

```

4784   \bool_if:NT \l_@@_parallelize_diags_bool
4785   {
4786     \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).

```

4787     \int_compare:nNnTF { \g_@@_ddots_int } = { \c_one_int }

```

If the diagonal line is the first one, we have no adjustment of the line to do but we store the Δ_x and the Δ_y of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```

4788     {
4789       \dim_gset:Nn \g_@@_delta_x_one_dim
4790       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4791       \dim_gset:Nn \g_@@_delta_y_one_dim
4792       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4793     }

```

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`.

```

4794     {
4795       \dim_compare:nNnF { \g_@@_delta_x_one_dim } = { \c_zero_dim }
4796       {
4797         \dim_set:Nn \l_@@_y_final_dim
4798         {
4799           \l_@@_y_initial_dim +
4800           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4801           \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4802         }
4803       }
4804     }
4805   }
4806   \@@_draw_line:
4807 }

```

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.

```

4808 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4809 {
4810   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4811   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4812   {
4813     \@@_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.

```

4814     \bool_if:NT \g_@@_aux_found_bool
4815     {
4816       \group_begin:
4817       \@@_open_shorten:
4818       \keys_set:nn { nicematrix / xdots } { #3 }
4819       \@@_color:o \l_@@_xdots_color_tl
4820       \@@_actually_draw_Iddots:
4821       \group_end:
4822     }
4823   }
4824 }

```

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`.

```

4825 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4826 {
4827   \bool_if:NTF \l_@@_initial_open_bool
4828   {
4829     \@@_open_y_initial_dim:
4830     \@@_open_x_initial_dim:
4831   }
4832   { \@@_set_initial_coords_from_anchor:n { south-west } }
4833   \bool_if:NTF \l_@@_final_open_bool
4834   {
4835     \@@_open_y_final_dim:
4836     \@@_open_x_final_dim:
4837   }
4838   { \@@_set_final_coords_from_anchor:n { north-east } }
4839   \bool_if:NT \l_@@_parallelize_diags_bool
4840   {
4841     \int_gincr:N \g_@@_iddots_int
4842     \int_compare:nNnTF { \g_@@_iddots_int } = { \c_one_int }
4843     {
4844       \dim_gset:Nn \g_@@_delta_x_two_dim
4845       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4846       \dim_gset:Nn \g_@@_delta_y_two_dim
4847       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4848     }
4849     {
4850       \dim_compare:nNnF { \g_@@_delta_x_two_dim } = { \c_zero_dim }
4851       {

```

```

4852         \dim_set:Nn \l_@@_y_final_dim
4853         {
4854             \l_@@_y_initial_dim +
4855             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4856             \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4857         }
4858     }
4859 }
4860 }
4861 \@@_draw_line:
4862 }

```

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`

```

4863 \cs_new_protected:Npn \@@_draw_line:
4864 {
4865     \pgfrememberpicturepositiononpagetrue
4866     \pgf@relevantforpicturesizefalse
4867     \bool_lazy_or:nnTF
4868     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4869     { \l_@@_dotted_bool }
4870     { \@@_draw_standard_dotted_line: }
4871     { \@@_draw_unstandard_dotted_line: }
4872 }

```

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.

```

4873 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4874 {
4875     \begin { scope }
4876     \@@_draw_unstandard_dotted_line:o
4877     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4878 }

```

We have used the fact that, in PGF, un color name can be put directly in a list of options (that's why we have put directly `\l_@@_xdots_color_tl`).

The argument of `\@@_draw_unstandard_dotted_line:n` is, in fact, the list of options.

```

4879 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4880 {
4881     \@@_draw_unstandard_dotted_line:nooo
4882     { #1 }
4883     \l_@@_xdots_up_tl
4884     \l_@@_xdots_down_tl
4885     \l_@@_xdots_middle_tl
4886 }
4887 \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.

```

4888 \hook_gput_code:nnn { begindocument } { . }
4889 {
4890   \IfPackageLoadedT { tikz }
4891   {
4892     \tikzset
4893     {
4894       @@_node_above / .style = { sloped , above } ,
4895       @@_node_below / .style = { sloped , below } ,
4896       @@_node_middle / .style =
4897       {
4898         sloped ,
4899         inner-sep = \c_@@_innersep_middle_dim
4900       }
4901     }
4902   }
4903 }

4904 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4905 {

```

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.

```

4906   \dim_zero_new:N \l_@@_l_dim
4907   \dim_set:Nn \l_@@_l_dim
4908   {
4909     \fp_to_dim:n
4910     {
4911       sqrt
4912       (
4913         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4914         +
4915         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4916       )
4917     }
4918   }

```

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.

```

4919   \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4920   {
4921     \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4922     \@@_draw_unstandard_dotted_line_i:
4923   }

```

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

```

4924   \bool_if:NT \l_@@_xdots_h_labels_bool
4925   {
4926     \tikzset
4927     {
4928       @@_node_above / .style = { auto = left } ,
4929       @@_node_below / .style = { auto = right } ,
4930       @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4931     }
4932   }
4933   \tl_if_empty:nF { #4 }
4934   { \tikzset { @@_node_middle / .append~style = { fill = white } } }

```

```

4935 \dim_zero:N \l_tmpa_dim
4936 \dim_zero:N \l_tmpb_dim
4937 \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
4938 {

```

We test whether the brace is vertical or horizontal.

```

4939 \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4940 { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
4941 { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
4942 }
4943 {
4944 \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
4945 {
4946 \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4947 { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
4948 { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
4949 }
4950 }
4951 \use:e
4952 {
4953 \exp_not:N \begin { scope }
4954 [ shift = {(\dim_use:N \l_tmpa_dim,\dim_use:N \l_tmpb_dim)} ]
4955 }
4956 \draw
4957 [ #1 ]
4958 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
4959 -- node [ @@_node_middle] { $ \scriptstyle #4 $ }
4960 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4961 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4962 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4963 \end { scope }
4964 \end { scope }
4965 }
4966 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4967 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4968 {
4969 \dim_set:Nn \l_tmpa_dim
4970 {
4971 \l_@@_x_initial_dim
4972 + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4973 * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4974 }
4975 \dim_set:Nn \l_tmpb_dim
4976 {
4977 \l_@@_y_initial_dim
4978 + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4979 * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4980 }
4981 \dim_set:Nn \l_@@_tmpc_dim
4982 {
4983 \l_@@_x_final_dim
4984 - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4985 * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4986 }
4987 \dim_set:Nn \l_@@_tmpd_dim
4988 {
4989 \l_@@_y_final_dim
4990 - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4991 * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4992 }
4993 \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4994 \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4995 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim

```

```

4996 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4997 }

```

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

```

4998 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4999 {
5000 \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.

```

5001 \dim_zero_new:N \l_@@_l_dim
5002 \dim_set:Nn \l_@@_l_dim
5003 {
5004 \fp_to_dim:n
5005 {
5006 sqrt
5007 (
5008 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5009 +
5010 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5011 )
5012 }
5013 }

```

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.

```

5014 \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
5015 {
5016 \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
5017 { \@@_draw_standard_dotted_line_i: }
5018 }
5019 \group_end:
5020 \bool_lazy_all:nF
5021 {
5022 { \tl_if_empty_p:N \l_@@_xdots_up_tl }
5023 { \tl_if_empty_p:N \l_@@_xdots_down_tl }
5024 { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
5025 }
5026 { \@@_labels_standard_dotted_line: }
5027 }
5028 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
5029 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
5030 {

```

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

```

5031 \int_set:Nn \l_tmpa_int
5032 {
5033 \dim_ratio:nn
5034 {
5035 \l_@@_l_dim
5036 - \l_@@_xdots_shorten_start_dim
5037 - \l_@@_xdots_shorten_end_dim
5038 }
5039 { \l_@@_xdots_inter_dim }
5040 }

```

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

```

5041 \dim_set:Nn \l_tmpa_dim

```

```

5042 {
5043   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5044   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5045 }
5046 \dim_set:Nn \l_tmpb_dim
5047 {
5048   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5049   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5050 }

```

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.

```

5051 \dim_gadd:Nn \l_@@_x_initial_dim
5052 {
5053   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5054   \dim_ratio:nn
5055   {
5056     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5057     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5058   }
5059   { 2 \l_@@_l_dim }
5060 }
5061 \dim_gadd:Nn \l_@@_y_initial_dim
5062 {
5063   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5064   \dim_ratio:nn
5065   {
5066     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5067     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5068   }
5069   { 2 \l_@@_l_dim }
5070 }
5071 \pgf@relevantforpicturesizefalse
5072 \int_step_inline:nnn { \c_zero_int } { \l_tmpa_int }
5073 {
5074   \pgfpathcircle
5075   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5076   { \l_@@_xdots_radius_dim }
5077   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
5078   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
5079 }
5080 \pgfusepathqfill
5081 }

```

```

5082 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
5083 {
5084   \pgfscope
5085   \pgftransformshift
5086   {
5087     \pgfpointlineattime { 0.5 }
5088     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5089     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
5090   }
5091   \fp_set:Nn \l_tmpa_fp
5092   {
5093     atand
5094     (
5095       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
5096       \l_@@_x_final_dim - \l_@@_x_initial_dim
5097     )
5098   }
5099   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
5100   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }

```



```

5101 \tl_if_empty:NF \l_@@_xdots_middle_tl
5102 {
5103   \begin { pgfscope }
5104   \pgfset { inner~sep = \c_@@_innersep_middle_dim }
5105   \pgfnode
5106     { rectangle }
5107     { center }
5108     {
5109       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5110       {
5111         $ % $
5112         \scriptstyle \l_@@_xdots_middle_tl
5113         $ % $
5114       }
5115     }
5116     { }
5117     {
5118       \pgfsetfillcolor { white }
5119       \pgfusepath { fill }
5120     }
5121   \end { pgfscope }
5122 }
5123 \tl_if_empty:NF \l_@@_xdots_up_tl
5124 {
5125   \pgfnode
5126     { rectangle }
5127     { south }
5128     {
5129       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5130       {
5131         $ % $
5132         \scriptstyle \l_@@_xdots_up_tl
5133         $ % $
5134       }
5135     }
5136     { }
5137     { \pgfusepath { } }
5138 }
5139 \tl_if_empty:NF \l_@@_xdots_down_tl
5140 {
5141   \pgfnode
5142     { rectangle }
5143     { north }
5144     {
5145       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5146       {
5147         $ % $
5148         \scriptstyle \l_@@_xdots_down_tl
5149         $ % $
5150       }
5151     }
5152     { }
5153     { \pgfusepath { } }
5154 }
5155 \endpgfscope
5156 }

```

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.

```
5157 \hook_gput_code:nnn { begindocument } { . }
5158 {
```

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

```
5159 \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
5160 \cs_new_protected:Npn \@@_Ldots:
5161 { \@@_collect_options:n { \@@_Ldots_i } }
5162 \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5163 {
5164   \int_if_zero:nTF { \c@jCol }
5165   { \@@_error:nn { in~first~col } { \Ldots } }
5166   {
5167     \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5168     { \@@_error:nn { in~last~col } { \Ldots } }
5169     {
5170       \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5171       { #1 , down = #2 , up = #3 , middle = #4 }
5172     }
5173   }
5174   \bool_if:NF \l_@@_nullify_dots_bool
5175   { \phantom { \ensuremath { \@@_old_ldots: } } }
5176   \bool_gset_true:N \g_@@_empty_cell_bool
5177 }
```

```
5178 \cs_new_protected:Npn \@@_Cdots:
5179 { \@@_collect_options:n { \@@_Cdots_i } }
5180 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5181 {
5182   \int_if_zero:nTF { \c@jCol }
5183   { \@@_error:nn { in~first~col } { \Cdots } }
5184   {
5185     \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5186     { \@@_error:nn { in~last~col } { \Cdots } }
5187     {
5188       \@@_instruction_of_type:nnn { \c_false_bool } { \Cdots }
5189       { #1 , down = #2 , up = #3 , middle = #4 }
5190     }
5191   }
5192   \bool_if:NF \l_@@_nullify_dots_bool
5193   { \phantom { \ensuremath { \@@_old_cdots: } } }
5194   \bool_gset_true:N \g_@@_empty_cell_bool
5195 }
```

```
5196 \cs_new_protected:Npn \@@_Vdots:
5197 { \@@_collect_options:n { \@@_Vdots_i } }
5198 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5199 {
5200   \int_if_zero:nTF { \c@iRow }
```

```

5201     { \@@_error:nn { in~first~row } { \Vdots } }
5202   {
5203     \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
5204       { \@@_error:nn { in~last~row } { \Vdots } }
5205     {
5206       \@@_instruction_of_type:nnn { \c_false_bool } { \Vdots }
5207       { #1 , down = #2 , up = #3 , middle = #4 }
5208     }
5209   }
5210   \bool_if:NF \l_@@_nullify_dots_bool
5211   { \phantom { \ensuremath { \@@_old_vdots: } } } }
5212   \bool_gset_true:N \g_@@_empty_cell_bool
5213 }

5214 \cs_new_protected:Npn \@@_Ddots:
5215 { \@@_collect_options:n { \@@_Ddots_i } }
5216 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5217 {
5218   \int_case:nnF \c@iRow
5219   {
5220     0 { \@@_error:nn { in~first~row } { \Ddots } }
5221     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5222   }
5223   {
5224     \int_case:nnF \c@jCol
5225     {
5226       0 { \@@_error:nn { in~first~col } { \Ddots } }
5227       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5228     }
5229     {
5230       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5231       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
5232       { #1 , down = #2 , up = #3 , middle = #4 }
5233     }
5234   }
5235 }
5236 \bool_if:NF \l_@@_nullify_dots_bool
5237 { \phantom { \ensuremath { \@@_old_ddots: } } } }
5238 \bool_gset_true:N \g_@@_empty_cell_bool
5239 }

5240 \cs_new_protected:Npn \@@_Iddots:
5241 { \@@_collect_options:n { \@@_Iddots_i } }
5242 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5243 {
5244   \int_case:nnF \c@iRow
5245   {
5246     0 { \@@_error:nn { in~first~row } { \Iddots } }
5247     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5248   }
5249   {
5250     \int_case:nnF \c@jCol
5251     {
5252       0 { \@@_error:nn { in~first~col } { \Iddots } }
5253       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5254     }
5255     {
5256       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5257       \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { \Iddots }
5258       { #1 , down = #2 , up = #3 , middle = #4 }
5259     }
5260   }

```

```

5261      \bool_if:NF \l_@@_nullify_dots_bool
5262      { \phantom { \ensuremath { \@@_old_iddots: } } }
5263      \bool_gset_true:N \g_@@_empty_cell_bool
5264    }
5265  }

```

End of the `\AddToHook`.

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

```

5266 \keys_define:nn { nicematrix / Ddots }
5267 {
5268   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5269   draw-first .default:n = true ,
5270   draw-first .value_forbidden:n = true
5271 }

```

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

```

5272 \cs_new_protected:Npn \@@_Hspace:
5273 {
5274   \bool_gset_true:N \g_@@_empty_cell_bool
5275   \hspace
5276 }

```

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`.

```

5277 \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`.

```

5278 \cs_new:Npn \@@_Hdotsfor:
5279 {
5280   \bool_lazy_and:nnTF
5281     { \int_if_zero_p:n { \c@jCol } }
5282     { \int_if_zero_p:n { \l_@@_first_col_int } }
5283     {
5284       \bool_if:NTF \g_@@_after_col_zero_bool
5285       {
5286         \multicolumn { 1 } { c } { }
5287         \@@_Hdotsfor_i:
5288       }
5289       { \@@_fatal:n { Hdotsfor~in~col~0 } }
5290     }
5291   {
5292     \multicolumn { 1 } { c } { }
5293     \@@_Hdotsfor_i:
5294   }
5295 }

```

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:`).

```

5296 \hook_gput_code:nnn { begindocument } { . }
5297 {

```

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

```

5298   \cs_new_protected:Npn \@@_Hdotsfor_i:
5299   { \@@_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`).

```

5300 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5301 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5302 {
5303   \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5304   {
5305     \@@_Hdotsfor:nnnn
5306     { \int_use:N \c@iRow }
5307     { \int_use:N \c@jCol }
5308     { #2 }
5309     {
5310       #1 , #3 ,
5311       down = \exp_not:n { #4 } ,
5312       up = \exp_not:n { #5 } ,
5313       middle = \exp_not:n { #6 }
5314     }
5315   }
5316   \prg_replicate:nn { #2 - 1 }
5317   {
5318     &
5319     \multicolumn { 1 } { c } { }
5320     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5321   }
5322 }
5323 }

```

```

5324 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5325 {
5326   \bool_set_false:N \l_@@_initial_open_bool
5327   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5328 \int_set:Nn \l_@@_initial_i_int { #1 }
5329 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5330 \int_compare:nNnTF { #2 } = { \c_one_int }
5331 {
5332   \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5333   \bool_set_true:N \l_@@_initial_open_bool
5334 }
5335 {
5336   \cs_if_exist:cTF
5337   {
5338     pgf @ sh @ ns @ \@@_env:
5339     - \int_use:N \l_@@_initial_i_int
5340     - \int_eval:n { #2 - 1 }
5341   }
5342   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5343   {
5344     \int_set:Nn \l_@@_initial_j_int { #2 }
5345     \bool_set_true:N \l_@@_initial_open_bool
5346   }
5347 }
5348 \int_compare:nNnTF { #2 + #3 - 1 } = { \c@jCol }
5349 {
5350   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5351   \bool_set_true:N \l_@@_final_open_bool
5352 }
5353 {
5354   \cs_if_exist:cTF
5355   {
5356     pgf @ sh @ ns @ \@@_env:

```

```

5357         - \int_use:N \l_@@_final_i_int
5358         - \int_eval:n { #2 + #3 }
5359     }
5360     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5361     {
5362         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5363         \bool_set_true:N \l_@@_final_open_bool
5364     }
5365 }
5366 \bool_if:NT \g_@@_aux_found_bool
5367 {
5368     \group_begin:
5369     \@@_open_shorten:
5370     \int_if_zero:nTF { #1 }
5371     { \color { nicematrix-first-row } }
5372     {
5373         \int_compare:nNnT { #1 } = { \g_@@_row_total_int }
5374         { \color { nicematrix-last-row } }
5375     }
5376     \keys_set:nn { nicematrix / xdots } { #4 }
5377     \@@_color:o \l_@@_xdots_color_tl
5378     \@@_actually_draw_Ldots:
5379     \group_end:
5380 }

```

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).

```

5381     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5382     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5383 }

```

```

5384 \hook_gput_code:nnn { begindocument } { . }
5385 {
5386     \cs_new_protected:Npn \@@_Vdotsfor:
5387     { \@@_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`).

```

5388     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5389     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5390     {
5391         \bool_gset_true:N \g_@@_empty_cell_bool
5392         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5393         {
5394             \@@_Vdotsfor:nnnn
5395             { \int_use:N \c@iRow }
5396             { \int_use:N \c@jCol }
5397             { #2 }
5398             {
5399                 #1 , #3 ,
5400                 down = \exp_not:n { #4 } ,
5401                 up = \exp_not:n { #5 } ,
5402                 middle = \exp_not:n { #6 }
5403             }
5404         }
5405     }
5406 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5407 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5408 {
5409   \bool_set_false:N \l_@@_initial_open_bool
5410   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5411   \int_set:Nn \l_@@_initial_j_int { #2 }
5412   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5413   \int_compare:nNnTF { #1 } = { \c_one_int }
5414   {
5415     \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5416     \bool_set_true:N \l_@@_initial_open_bool
5417   }
5418   {
5419     \cs_if_exist:cTF
5420     {
5421       pgf @ sh @ ns @ \@@_env:
5422       - \int_eval:n { #1 - 1 }
5423       - \int_use:N \l_@@_initial_j_int
5424     }
5425     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5426     {
5427       \int_set:Nn \l_@@_initial_i_int { #1 }
5428       \bool_set_true:N \l_@@_initial_open_bool
5429     }
5430   }
5431   \int_compare:nNnTF { #1 + #3 - 1 } = { \c_iRow }
5432   {
5433     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5434     \bool_set_true:N \l_@@_final_open_bool
5435   }
5436   {
5437     \cs_if_exist:cTF
5438     {
5439       pgf @ sh @ ns @ \@@_env:
5440       - \int_eval:n { #1 + #3 }
5441       - \int_use:N \l_@@_final_j_int
5442     }
5443     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5444     {
5445       \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5446       \bool_set_true:N \l_@@_final_open_bool
5447     }
5448   }
5449   \bool_if:NT \g_@@_aux_found_bool
5450   {
5451     \group_begin:
5452     \@@_open_shorten:
5453     \int_if_zero:nTF { #2 }
5454     { \color { nicematrix-first-col } }
5455     {
5456       \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
5457       { \color { nicematrix-last-col } }
5458     }
5459     \keys_set:nn { nicematrix / xdots } { #4 }
5460     \@@_color:o \l_@@_xdots_color_tl
5461     \bool_if:NTF \l_@@_Vbrace_bool
5462     { \@@_actually_draw_Vbrace: }
5463     { \@@_actually_draw_Vdots: }
5464     \group_end:
5465   }

```

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).

```

5466 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5467 { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5468 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5469 \NewDocumentCommand \@@_rotate: { 0 { } }
5470 {
5471   \bool_gset_true:N \g_@@_rotate_bool
5472   \keys_set:nn { nicematrix / rotate } { #1 }
5473   \ignorespaces
5474 }

```

The command `\@@_rotate_p_col:` will be linked to `\rotate` in the the cells of the columns of type *p* and *al*.

```

5475 \cs_new_protected:Npn \@@_rotate_p_col: { \@@_error:n { rotate~in~p~col } }

5476 \keys_define:nn { nicematrix / rotate }
5477 {
5478   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5479   c .value_forbidden:n = true ,
5480   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5481 }

```

19 The command `\line` accessible in code-after

In the `\CodeAfter`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format *i-j*) and draws a dotted line between these cells. In fact, it also works with names of blocks.

First, we write a command with the following behaviour:

- If the argument is of the format *i-j*, our command applies the command `\int_eval:n` to *i* and *j* ;
- If not (that is to say, when it’s a name of a `\Block`), the argument is left unchanged.

This must *not* be protected (and is, of course fully expandable).¹⁴

```

5482 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5483 {
5484   \tl_if_empty:nTF { #2 }
5485   { #1 }
5486   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5487 }
5488 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5489 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

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

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).

```
5490 \hook_gput_code:nnn { begindocument } { . }
5491 {
```

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

```
5492 \tl_set_rescan:Nnn \l_tmpa_tl { }
5493 { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5494 \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5495 {
5496   \group_begin:
5497   \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5498   \@@_color:o \l_@@_xdots_color_tl
5499   \use:e
5500   {
5501     \@@_line_i:nn
5502     { \@@_double_int_eval:n #2 - \q_stop }
5503     { \@@_double_int_eval:n #3 - \q_stop }
5504   }
5505   \group_end:
5506 }
5507 }

5508 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5509 {
5510   \bool_set_false:N \l_@@_initial_open_bool
5511   \bool_set_false:N \l_@@_final_open_bool
5512   \bool_lazy_or:nnTF
5513   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5514   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5515   { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }
```

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

```
5516 { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5517 }

5518 \hook_gput_code:nnn { begindocument } { . }
5519 {
5520   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5521   {
```

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:`.

```
5522 \c_@@_pgfortikzpicture_tl
5523 \@@_draw_line_iii:nn { #1 } { #2 }
5524 \c_@@_endpgfortikzpicture_tl
5525 }
5526 }
```

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

```
5527 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5528 {
5529   \pgfrememberpicturepositiononpagetrue
5530   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5531   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5532   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5533   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5534   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5535   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5536   \@@_draw_line:
5537 }
```

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.

```
5538 \cs_new:Npn @@_if_row_less_than:nn { \int_compare:nNt { \c@iRow } < }
5539 \cs_new:Npn @@_if_col_greater_than:nn { \int_compare:nNf { \c@jCol } < }
```

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

```
5540 \cs_set_protected:Npn @@_put_in_row_style:n #1
5541 {
5542   \tl_gput_right:Ne \g_@@_row_style_tl
5543   {
```

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

```
5544   \exp_not:N
5545   @@_if_row_less_than:nn
5546   { \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`).

```
5547   {
5548     \exp_not:N
5549     @@_if_col_greater_than:nn
5550     { \int_eval:n { \c@jCol } }
5551     { \exp_not:n { #1 } \scan_stop: }
5552   }
5553 }
5554 }
5555 \cs_generate_variant:Nn @@_put_in_row_style:n { e }
```

```
5556 \keys_define:nn { nicematrix / RowStyle }
5557 {
5558   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5559   cell-space-top-limit .value_required:n = true ,
5560   cell-space-top-limit+ .code:n =
5561     \dim_set:Nn \l_tmpa_dim { \l_@@_cell_space_top_limit_dim + #1 } ,
5562   cell-space-top-limit+ .value_required:n = true ,
5563   cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
5564   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5565   cell-space-bottom-limit .value_required:n = true ,
5566   cell-space-bottom-limit+ .code:n =
5567     \dim_set:Nn \l_tmpb_dim { \l_@@_cell_space_bottom_limit_dim + #1 } ,
5568   cell-space-bottom-limit+ .value_required:n = true ,
5569   cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
5570   cell-space-limits .meta:n =
```

```

5571     {
5572         cell-space-top-limit = #1 ,
5573         cell-space-bottom-limit = #1 ,
5574     } ,
5575     cell-space-limits+ .meta:n =
5576     {
5577         cell-space-top-limit += #1 ,
5578         cell-space-bottom-limit += #1 ,
5579     } ,
5580     cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
5581     color .tl_set:N = \l_@@_color_tl ,
5582     color .value_required:n = true ,
5583     bold .bool_set:N = \l_@@_bold_row_style_bool ,
5584     bold .default:n = true ,
5585     nb-rows .code:n =
5586         \str_if_eq:eeTF { #1 } { * }
5587         { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5588         { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5589     nb-rows .value_required:n = true ,
5590     fill .tl_set:N = \l_@@_fill_tl ,
5591     fill .value_required:n = true ,

```

In fine, the opacity will be applied by `\pgfsetfillopacity`.

```

5592     opacity .tl_set:N = \l_@@_opacity_tl ,
5593     opacity .value_required:n = true ,
5594     rowcolor .tl_set:N = \l_@@_fill_tl ,
5595     rowcolor .value_required:n = true ,
5596     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5597     rounded-corners .default:n = 4 pt ,
5598     unknown .code:n =
5599         \@@_unknown_key:nn
5600         { nicematrix / RowStyle }
5601         { Unknown-key-for-RowStyle }
5602 }

```

```

5603 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5604 {
5605     \group_begin:
5606     \tl_clear:N \l_@@_fill_tl
5607     \tl_clear:N \l_@@_opacity_tl
5608     \tl_clear:N \l_@@_color_tl
5609     \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5610     \dim_zero:N \l_@@_rounded_corners_dim
5611     \dim_zero:N \l_tmpa_dim
5612     \dim_zero:N \l_tmpb_dim
5613     \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5614     \tl_if_empty:NF \l_@@_fill_tl
5615     {
5616         \@@_add_opacity_to_fill:
5617         \tl_gput_right:Ne \g_@@_pre_code_before_tl
5618         {

```

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

```

5619         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5620         { \int_use:N \c_iRow - \int_use:N \c_jCol }
5621         {
5622             \int_eval:n { \c_iRow + \l_@@_key_nb_rows_int - 1 }
5623             - *
5624         }
5625         { \dim_use:N \l_@@_rounded_corners_dim }
5626     }
5627 }

```

```

5628 \@@_put_in_row_style:n { \exp_not:n { #2 } }
\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.
5629 \dim_compare:nNtT { \l_tmpa_dim } > { \c_zero_dim }
5630 {
5631 \@@_put_in_row_style:e
5632 {
5633 \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5634 {

```

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

```

5635 \dim_set:Nn \l_@@_cell_space_top_limit_dim
5636 { \dim_use:N \l_tmpa_dim }
5637 }
5638 }
5639 }

```

`\l_tmpb_dim` is the value of the key cell-space-bottom-limit of `\RowStyle`.

```

5640 \dim_compare:nNtT { \l_tmpb_dim } > { \c_zero_dim }
5641 {
5642 \@@_put_in_row_style:e
5643 {
5644 \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5645 {
5646 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5647 { \dim_use:N \l_tmpb_dim }
5648 }
5649 }
5650 }

```

`\l_@@_color_tl` is the value of the key color of `\RowStyle`.

```

5651 \tl_if_empty:NF \l_@@_color_tl
5652 {
5653 \@@_put_in_row_style:e
5654 {
5655 \mode_leave_vertical:
5656 \@@_color:n { \l_@@_color_tl }
5657 }
5658 }

```

`\l_@@_bold_row_style_bool` is the value of the key bold.

```

5659 \bool_if:NT \l_@@_bold_row_style_bool
5660 {
5661 \@@_put_in_row_style:n
5662 {
5663 \exp_not:n
5664 {
5665 \if_mode_math:
5666 $ % $
5667 \bfseries \boldmath
5668 $ % $
5669 \else:
5670 \bfseries \boldmath
5671 \fi:
5672 }
5673 }
5674 }
5675 \group_end:
5676 \g_@@_row_style_tl
5677 \ignorespaces
5678 }

```

The following commande must *not* be protected.

```

5679 \cs_new:Npn \@@_rounded_from_row:n #1
5680 {
5681 \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

In the following code, the “- 1” is *not* a subtraction.

```

5682     { \int_eval:n { #1 } - 1 }
5683     {
5684         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5685         - \exp_not:n { \int_use:N \c@jCol }
5686     }
5687     { \dim_use:N \l_@@_rounded_corners_dim }
5688 }

```

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).

```

5689 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5690 {

```

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.

```

5691     \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`.

```

5692     \str_if_in:nnF { #1 } { !! }
5693     {
5694         \seq_map_indexed_inline:Nn \g_@@_colors_seq

```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```

5695         { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5696     }
5697     \int_if_zero:nTF { \l_tmpa_int }

```

First, the case where the color is a *new* color (not in the sequence).

```

5698     {
5699         \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5700         \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5701     }

```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```

5702     { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5703   }
5704   \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5705   \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }

```

The following command must be used within a `\pgfpicture`.

```

5706   \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5707   {
5708     \dim_compare:nNtT { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim }
5709     {

```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5710     \group_begin:
5711     \pgfsetcornersarced
5712     {
5713       \pgfpoint
5714       { \l_@@_tab_rounded_corners_dim }
5715       { \l_@@_tab_rounded_corners_dim }
5716     }

```

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.

```

5717     \bool_if:NTF \l_@@_hvlines_bool
5718     {
5719       \pgfpathrectanglecorners
5720       {
5721         \pgfpointadd
5722         { \@@_qpoint:n { row-1 } }
5723         { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5724       }
5725       {
5726         \pgfpointadd
5727         {
5728           \@@_qpoint:n
5729           { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5730         }
5731         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5732       }
5733     }
5734     {
5735       \pgfpathrectanglecorners
5736       { \@@_qpoint:n { row-1 } }
5737       {
5738         \pgfpointadd
5739         {
5740           \@@_qpoint:n
5741           { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5742         }
5743         { \pgfpoint \c_zero_dim \arrayrulewidth }
5744       }
5745     }
5746     \pgfusepath { clip }
5747     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5748   }
5749 }

```

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`).

```

5750 \cs_new_protected:Npn \@@_actually_color:
5751 {
5752   \pgfpicture
5753   \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.

```

5754   \@@_clip_with_rounded_corners:
5755   \seq_map_indexed_inline:Nn \g_@@_colors_seq
5756   {
5757     \int_compare:nNnTF { ##1 } = { \c_one_int }
5758     {
5759       \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5760       \use:c { g_@@_color _ 1 _tl }
5761       \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5762     }
5763     {
5764       \begin { pgfscope }
5765         \@@_color_opacity: ##2
5766         \use:c { g_@@_color _ ##1 _tl }
5767         \tl_gclear:c { g_@@_color _ ##1 _tl }
5768         \pgfusepath { fill }
5769       \end { pgfscope }
5770     }
5771   }
5772   \endpgfpicture
5773 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5774 \cs_new_protected:Npn \@@_color_opacity:
5775 {
5776   \peek_meaning:NTF [
5777     { \@@_color_opacity:w }
5778     { \@@_color_opacity:w [ ] }
5779   }

```

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.

```

5780 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5781 {
5782   \tl_clear:N \l_tmpa_tl
5783   \keys_set_known:nn { nicematrix / color-opacity } { #1 } \l_tmpb_tl

```

`\l_tmpa_tl` (if not empty) is now the opacity and `\l_tmpb_tl` (if not empty) is now the colorimetric space.

```

5784   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5785   \tl_if_empty:NTF \l_tmpb_tl
5786     { \@declaredcolor }
5787     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5788 }

```

The following set of keys is used by the command `\@@_color_opacity:w`.

```

5789 \keys_define:nn { nicematrix / color-opacity }
5790 {
5791   opacity .tl_set:N          = \l_tmpa_tl ,
5792   opacity .value_required:n = true
5793 }

```

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

```

5794 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5795 {
5796   \def \l_@@_rows_tl { #1 }
5797   \def \l_@@_cols_tl { #2 }
5798   \@@_cartesian_path:
5799 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5800 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5801 {
5802   \tl_if_blank:nF { #2 }
5803   {
5804     \@@_add_to_colors_seq:en
5805     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5806     { \@@_cartesian_color:nn { #3 } { - } }
5807   }
5808 }

```

Here an example: `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5809 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5810 {
5811   \tl_if_blank:nF { #2 }
5812   {
5813     \@@_add_to_colors_seq:en
5814     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5815     { \@@_cartesian_color:nn { - } { #3 } }
5816   }
5817 }

```

Here is an example: `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5818 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5819 {
5820   \tl_if_blank:nF { #2 }
5821   {
5822     \@@_add_to_colors_seq:en
5823     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5824     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5825   }
5826 }

```

The last argument is the radius of the corners of the rectangle.

```

5827 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5828 {
5829   \tl_if_blank:nF { #2 }
5830   {
5831     \@@_add_to_colors_seq:en
5832     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5833     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5834   }
5835 }

```

The last argument is the radius of the corners of the rectangle.

```

5836 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5837 {
5838   \@@_cut_on_hyphen:w #1 \q_stop
5839   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5840   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5841   \@@_cut_on_hyphen:w #2 \q_stop
5842   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5843   \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`.

```
5844 \@@_cartesian_path:n { #3 }
5845 }
```

Here is an example: `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```
5846 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5847 {
5848   \clist_map_inline:nn { #3 }
5849   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5850 }
```

```
5851 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5852 {
5853   \int_step_inline:nn { \c@iRow }
5854   {
5855     \int_step_inline:nn { \c@jCol }
5856     {
5857       \int_if_even:nTF { ####1 + ##1 }
5858       { \@@_cellcolor [ #1 ] { #2 } }
5859       { \@@_cellcolor [ #1 ] { #3 } }
5860       { ##1 - ####1 }
5861     }
5862   }
5863 }
```

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”.

```
5864 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5865 {
5866   \@@_rectanglecolor [ #1 ] { #2 }
5867   { 1 - 1 }
5868   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5869 }
```

```
5870 \keys_define:nn { nicematrix / rowcolors }
5871 {
5872   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5873   respect-blocks .default:n = true ,
5874   cols .tl_set:N = \l_@@_cols_tl ,
5875   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5876   restart .default:n = true ,
5877   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5878 }
```

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*.

```
5879 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5880 {
```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5881 \group_begin:
5882 \seq_clear_new:N \l_@@_colors_seq
5883 \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5884 \tl_clear_new:N \l_@@_cols_tl
5885 \tl_set:Nn \l_@@_cols_tl { - }
5886 \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).

```

5887 \int_zero_new:N \l_@@_color_int
5888 \int_set_eq:NN \l_@@_color_int \c_one_int
5889 \bool_if:NT \l_@@_respect_blocks_bool
5890 {

```

We don't want to take into account a block which is completely in the “first column” (number 0) or in the “last column” and that's why we filter the sequence of the blocks (in the sequence `\l_tmpa_seq`).

```

5891 \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5892 \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5893 { \@@_not_in_exterior_p:nnnnn ##1 }
5894 }
5895 \pgfpicture
5896 \pgf@relevantforpicturesizefalse

```

`#2` is the list of intervals of rows.

```

5897 \clist_map_inline:nn { #2 }
5898 {
5899 \tl_set:Nn \l_tmpa_tl { ##1 }
5900 \tl_if_in:NnTF \l_tmpa_tl { - }
5901 { \@@_cut_on_hyphen:w ##1 \q_stop }
5902 { \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.

```

5903 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5904 \int_set:Nn \l_@@_color_int
5905 { \bool_if:NNTF \l_@@_rowcolors_restart_bool { 1 } { \l_tmpa_tl } }
5906 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5907 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5908 {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5909 \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).

```

5910 \bool_if:NT \l_@@_respect_blocks_bool
5911 {
5912 \seq_set_filter:Nnn \l_tmpb_seq \l_tmpa_seq
5913 { \@@_intersect_our_row_p:nnnnn #####1 }
5914 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5915 }
5916 \tl_set:Nn \l_@@_rows_tl
5917 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5918 \tl_set:Nn \l_@@_color_tl
5919 {
5920 \@@_color_index:n
5921 {
5922 \int_mod:nn
5923 { \l_@@_color_int - 1 }
5924 { \seq_count:N \l_@@_colors_seq }
5925 + 1
5926 }

```

```

5927     }
5928     \tl_if_empty:NF \l_@@_color_tl
5929     {
5930         \@@_add_to_colors_seq:ee
5931         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5932         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5933     }
5934     \int_incr:N \l_@@_color_int
5935     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5936 }
5937 }
5938 \endpgfpicture
5939 \group_end:
5940 }

```

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.

```

5941 \cs_new:Npn \@@_color_index:n #1
5942 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5943     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5944     { \@@_color_index:n { #1 - 1 } }
5945     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5946 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by curryfication.

```

5947 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5948 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5949 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5950 {
5951     \int_compare:nNnT { #3 } > { \l_tmpb_int }
5952     { \int_set:Nn \l_tmpb_int { #3 } }
5953 }

```

```

5954 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5955 {
5956     \int_if_zero:nTF { #4 }
5957     { \prg_return_false: }
5958     {
5959         \int_compare:nNnTF { #2 } > { \c@jCol }
5960         { \prg_return_false: }
5961         { \prg_return_true: }
5962     }
5963 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5964 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5965 {
5966     \int_compare:nNnTF { #1 } > { \l_tmpa_int }
5967     { \prg_return_false: }
5968     {
5969         \int_compare:nNnTF { \l_tmpa_int } > { #3 }
5970         { \prg_return_false: }
5971         { \prg_return_true: }
5972     }
5973 }

```

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`).

```

5974 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5975 {
5976   \dim_compare:nNnTF { #1 } = { \c_zero_dim }
5977   {
5978     \bool_if:NTF \l_@@_nocolor_used_bool
5979     { \@@_cartesian_path_normal_ii: }
5980     {
5981       \clist_if_empty:NTF \l_@@_corners_cells_clist
5982       { \@@_cartesian_path_normal_i:n { #1 } }
5983       { \@@_cartesian_path_normal_ii: }
5984     }
5985   }
5986   { \@@_cartesian_path_normal_i:n { #1 } }
5987 }

```

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.

```

5988 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5989 {
5990   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5991   \clist_map_inline:Nn \l_@@_cols_tl
5992   {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

5993     \def \l_tmpa_tl { ##1 }
5994     \tl_if_in:NnTF \l_tmpa_tl { - }
5995     { \@@_cut_on_hyphen:w ##1 \q_stop }
5996     { \def \l_tmpb_tl { ##1 } }
5997     \tl_if_empty:NTF \l_tmpa_tl
5998     { \def \l_tmpa_tl { 1 } }
5999     {
6000       \str_if_eq:eeT { \l_tmpa_tl } { * }
6001       { \def \l_tmpa_tl { 1 } }
6002     }
6003     \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_col_total_int }
6004     { \@@_error:n { Invalid~col~number } }
6005     \tl_if_empty:NTF \l_tmpb_tl
6006     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6007     {
6008       \str_if_eq:eeT { \l_tmpb_tl } { * }
6009       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6010     }
6011     \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_col_total_int }
6012     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

6013     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6014     \@@_qpoint:n { col - \l_tmpa_tl }
6015     \int_compare:nNnTF { \l_@@_first_col_int } = { \l_tmpa_tl }
6016     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6017     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6018     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
6019     \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.

```

6020 \clist_map_inline:Nn \l_@@_rows_tl
6021 {
6022   \def \l_tmpa_tl { #####1 }
6023   \tl_if_in:NnTF \l_tmpa_tl { - }
6024     { \@@_cut_on_hyphen:w #####1 \q_stop }
6025     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
6026   \tl_if_empty:NnTF \l_tmpa_tl
6027     { \def \l_tmpa_tl { 1 } }
6028     {
6029       \str_if_eq:eeT { \l_tmpa_tl } { * }
6030       { \def \l_tmpa_tl { 1 } }
6031     }
6032   \tl_if_empty:NnTF \l_tmpb_tl
6033     { \tl_set:Nn \l_tmpb_tl { \int_use:N \c@iRow } }
6034     {
6035       \str_if_eq:eeT { \l_tmpb_tl } { * }
6036       { \tl_set:Nn \l_tmpb_tl { \int_use:N \c@iRow } }
6037     }
6038   \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_row_total_int }
6039     { \@@_error:n { Invalid-row-number } }
6040   \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_row_total_int }
6041     { \tl_set:Nn \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`.

```

6042 \cs_if_exist:cF
6043 { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
6044 {
6045   \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
6046   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6047   \@@_qpoint:n { row - \l_tmpa_tl }
6048   \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6049   \pgfpathrectanglecorners
6050     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6051     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6052   }
6053 }
6054 }
6055 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

6056 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
6057 {
6058   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6059   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6060 \clist_map_inline:Nn \l_@@_cols_tl
6061 {
6062   \@@_qpoint:n { col - ##1 }
6063   \int_compare:nNnTF { \l_@@_first_col_int } = { ##1 }
6064     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6065     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6066   \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
6067   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

6068 \clist_map_inline:Nn \l_@@_rows_tl
6069 {
6070   \@@_if_in_corner:nF { #####1 - ##1 }
6071   {
6072     \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
6073     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6074     \@@_qpoint:n { row - #####1 }

```

```

6075         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6076         \cs_if_exist:cF { @@ _ nocolor _ #####1 - #1 }
6077         {
6078             \pgfpathrectanglecorners
6079             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6080             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6081         }
6082     }
6083 }
6084 }
6085 }

```

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`).

```

6086 \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

```

6087 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
6088 {
6089     \bool_set_true:N \l_@@_nocolor_used_bool
6090     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6091     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6092     \clist_map_inline:Nn \l_@@_rows_tl
6093     {
6094         \clist_map_inline:Nn \l_@@_cols_tl
6095         { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
6096     }
6097 }

```

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.

```

6098 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
6099 {
6100     \clist_set_eq:NN \l_tmpa_clist #1
6101     \clist_clear:N #1
6102     \clist_map_inline:Nn \l_tmpa_clist
6103     {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

6104         \def \l_tmpa_tl { ##1 }
6105         \tl_if_in:NnTF \l_tmpa_tl { - }
6106         { \@@_cut_on_hyphen:w ##1 \q_stop }
6107         { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
6108         \bool_lazy_or:nnT
6109         { \str_if_eq_p:ee { \l_tmpa_tl } { * } }
6110         { \tl_if_blank_p:o \l_tmpa_tl }
6111         { \def \l_tmpa_tl { 1 } }
6112         \bool_lazy_or:nnT
6113         { \str_if_eq_p:ee { \l_tmpb_tl } { * } }
6114         { \tl_if_blank_p:o \l_tmpb_tl }
6115         { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6116         \int_compare:nNnT { \l_tmpb_tl } > { #2 }
6117         { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6118         \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
6119         { \clist_put_right:Nn #1 { #####1 } }
6120     }
6121 }

```

The following command will be linked to `\cellcolor` in the tabular.

```
6122 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
6123 {
6124   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6125   {
```

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`).

```
6126     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
6127     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6128   }
6129   \ignorespaces
6130 }

6131 \NewDocumentCommand \@@_cellcolor_error { 0 { } m }
6132 { \@@_error:n { cellcolor~in~Block } }
6133 % \end{macrocode}
6134 %
6135 % \begin{macrocode}
6136 \NewDocumentCommand \@@_rowcolor_error { 0 { } m }
6137 { \@@_error:n { rowcolor~in~Block } }
6138 % \end{macrocode}
6139 %
6140 % \bigskip
6141 % The following command will be linked to |\rowcolor| in the tabular.
6142 % \begin{macrocode}
6143 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
6144 {
6145   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6146   {
6147     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
6148     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6149     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
6150   }
6151   \ignorespaces
6152 }
```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```
6153 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
6154 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }
```

The braces around `#2` and `#3` are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```
6155 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
6156 {
```

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`.

```
6157   \seq_gclear:N \g_tmpa_seq
6158   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6159   { \@@_rowlistcolors_tabular:nnnn #1 }
6160   \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`.

```
6161   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
6162   {
```

```

6163     { \int_use:N \c@iRow }
6164     { \exp_not:n { #1 } }
6165     { \exp_not:n { #2 } }
6166     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
6167   }
6168   \ignorespaces
6169 }

```

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`).

```

6170 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
6171 {
6172   \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).

```

6173   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
6174   {
6175     \tl_gput_right:Ne \g_@@_pre_code_before_tl
6176     {
6177       \@@_rowlistcolors
6178       [ \exp_not:n { #2 } ]
6179       { #1 - \int_eval:n { \c@iRow - 1 } }
6180       { \exp_not:n { #3 } }
6181       [ \exp_not:n { #4 } ]
6182     }
6183   }
6184 }

```

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.

```

6185 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
6186 {
6187   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6188   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
6189   \seq_gclear:N \g_@@_rowlistcolors_seq
6190 }
6191 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6192 {
6193   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6194   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6195 }

```

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.

```

6196 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
6197 {

```

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).

```

6198   \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
6199   {

```


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).

```

6200     \tl_gput_left:N\g_@@_pre_code_before_tl
6201     {
6202         \exp_not:N \columncolor [ #1 ]
6203         { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6204     }
6205 }
6206 }

6207 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6208 {
6209     \clist_map_inline:nn { #1 }
6210     {
6211         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6212         { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6213         \columncolor { nocolor } { ##1 }
6214     }
6215 }

6216 \cs_new_protected:Npn \@@_EmptyRow:n #1
6217 {
6218     \clist_map_inline:nn { #1 }
6219     {
6220         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6221         { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6222         \rowcolor { nocolor } { ##1 }
6223     }
6224 }

```

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`.

```

6225 \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`.

```

6226 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6227 {
6228     \int_if_zero:nTF { \l_@@_first_col_int }
6229     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6230     {
6231         \int_if_zero:nTF { \c@jCol }
6232         {
6233             \int_compare:nNnF { \c@iRow } = { -1 }
6234             {

```

```

6235         \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int - 1 }
6236         { #1 }
6237     }
6238 }
6239 { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6240 }
6241 }

```

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.

```

6242 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6243 {
6244     \int_if_zero:nF { \c@iRow }
6245     {
6246         \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
6247         {
6248             \int_compare:nNnT { \c@jCol } > { \c_zero_int }
6249             { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6250         }
6251     }
6252 }

```

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`.

```

6253 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6254 {
6255     \IfPackageLoadedTF { tikz }
6256     {
6257         \IfPackageLoadedTF { booktabs }
6258         { #2 }
6259         { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6260     }
6261     { \@@_error:nn { TopRule~without~tikz } { #1 } }
6262 }
6263 \NewExpandableDocumentCommand { \@@_TopRule } { }
6264 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
6265 \cs_new:Npn \@@_TopRule_i:
6266 {
6267     \noalign \bgroup
6268     \peek_meaning:NTF [
6269     { \@@_TopRule_ii: }
6270     { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6271 }
6272 \NewDocumentCommand \@@_TopRule_ii: { o }
6273 {
6274     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6275     {
6276         \@@_hline:n
6277         {
6278             position = \int_eval:n { \c@iRow + 1 } ,
6279             tikz =
6280             {
6281                 line-width = #1 ,
6282                 yshift = 0.25 \arrayrulewidth ,
6283                 shorten~< = - 0.5 \arrayrulewidth
6284             } ,
6285             total-width = #1
6286         }

```

```

6287     }
6288     \skip_vertical:n { \belowrulesep + #1 }
6289     \egroup
6290 }

6291 \NewExpandableDocumentCommand { \@@_BottomRule } { } {
6292   { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }

6293   \cs_new:Npn \@@_BottomRule_i:
6294     {
6295       \noalign \bgroup
6296       \peek_meaning:NTF [
6297         { \@@_BottomRule_ii: }
6298         { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6299     }

6300 \NewDocumentCommand \@@_BottomRule_ii: { o }
6301 {
6302   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6303   {
6304     \@@_hline:n
6305     {
6306       position = \int_eval:n { \c@iRow + 1 } ,
6307       tikz =
6308       {
6309         line-width = #1 ,
6310         yshift = 0.25 \arrayrulewidth ,
6311         shorten-< = - 0.5 \arrayrulewidth
6312       } ,
6313       total-width = #1 ,
6314     }
6315   }
6316   \skip_vertical:N \aboverulesep
6317   \@@_create_row_node_i:
6318   \skip_vertical:n { #1 }
6319   \egroup
6320 }

6321 \NewExpandableDocumentCommand { \@@_MidRule } { } {
6322   { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }

6323   \cs_new:Npn \@@_MidRule_i:
6324     {
6325       \noalign \bgroup
6326       \peek_meaning:NTF [
6327         { \@@_MidRule_ii: }
6328         { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6329     }

6330 \NewDocumentCommand \@@_MidRule_ii: { o }
6331 {
6332   \skip_vertical:N \aboverulesep
6333   \@@_create_row_node_i:
6334   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6335   {
6336     \@@_hline:n
6337     {
6338       position = \int_eval:n { \c@iRow + 1 } ,
6339       tikz =
6340       {
6341         line-width = #1 ,
6342         yshift = 0.25 \arrayrulewidth ,
6343         shorten-< = - 0.5 \arrayrulewidth
6344       } ,
6345       total-width = #1 ,
6346     }
6347   }

```

```

6348     \skip_vertical:n { \belowrulesep + #1 }
6349     \egroup
6350 }

```

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.

```

6351 \keys_define:nn { nicematrix / Rules }
6352 {
6353     position .int_set:N = \l_@@_position_int ,
6354     position .value_required:n = true ,
6355     start .int_set:N = \l_@@_start_int ,
6356     end .code:n =
6357         \bool_lazy_or:nnTF
6358         { \tl_if_empty_p:n { #1 } }
6359         { \str_if_eq_p:ee { #1 } { last } }
6360         { \int_set_eq:NN \l_@@_end_int \c@jCol }
6361         { \int_set:Nn \l_@@_end_int { #1 } }
6362 }

```

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.

```

6363 \keys_define:nn { nicematrix / RulesBis }
6364 {
6365     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6366     multiplicity .initial:n = 1 ,
6367     dotted .bool_set:N = \l_@@_dotted_bool ,
6368     dotted .initial:n = false ,
6369     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]`.

```

6370     color .code:n =
6371         \@@_set_CTarc:n { #1 }
6372         \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6373     color .value_required:n = true ,
6374     sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6375     sep-color .value_required:n = true ,

```

If the user uses the key `tikz`, the rule (or more precisely: the different sub-rules since a rule may be broken by blocks or others) will be drawn with TikZ.

```

6376     tikz .code:n =
6377         \IfPackageLoadedTF { tikz }
6378         { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6379         { \@@_error:n { tikz-without-tikz } } ,
6380     tikz .value_required:n = true ,
6381     total-width .dim_set:N = \l_@@_rule_width_dim ,
6382     total-width .value_required:n = true ,
6383     width .meta:n = { total-width = #1 } ,
6384     unknown .code:n =
6385         \@@_unknown_key:nn

```

```

6386         { nicematrix / RulesBis }
6387         { Unknown-key-for-RulesBis }
6388     }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6389 \cs_new_protected:Npn \@@_vline:n #1
6390 {

```

The group is for the options.

```

6391     \group_begin:
6392     \int_set_eq:NN \l_@@_end_int \c@iRow
6393     \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).

```

6394     \int_compare:nNnT { \l_@@_position_int } < { \c@jCol + 2 }
6395     \@@_vline_i:
6396     \group_end:
6397 }

```

```

6398 \cs_new_protected:Npn \@@_vline_i:
6399 {

```

`\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`.

```

6400     \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
6401     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6402     \l_tmpa_tl
6403     {

```

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.

```

6404         \bool_gset_true:N \g_tmpa_bool
6405         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6406             { \@@_test_vline_in_block:nnnnn ##1 }
6407         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6408             { \@@_test_vline_in_block:nnnnn ##1 }
6409         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6410             { \@@_test_vline_in_stroken_block:nnnn ##1 }
6411         \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_v: }
6412         \bool_if:NTF \g_tmpa_bool
6413             {
6414                 \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.

```

6415             { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6416         }
6417         {
6418             \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6419             {
6420                 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6421                 \@@_vline_ii:
6422                 \int_zero:N \l_@@_local_start_int
6423             }
6424         }
6425     }
6426     \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6427     {

```

```

6428         \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6429         \@@_vline_ii:
6430     }
6431 }

6432 \cs_new_protected:Npn \@@_test_in_corner_v:
6433 {
6434     \int_compare:nNnTF { \l_tmpb_tl } = { \c@jCol + 1 }
6435     {
6436         \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6437         { \bool_set_false:N \g_tmpa_bool }
6438     }
6439     {
6440         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6441         {
6442             \int_compare:nNnTF { \l_tmpb_tl } = { \c_one_int }
6443             { \bool_set_false:N \g_tmpa_bool }
6444             {
6445                 \@@_if_in_corner:nT
6446                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6447                 { \bool_set_false:N \g_tmpa_bool }
6448             }
6449         }
6450     }
6451 }

6452 \cs_new_protected:Npn \@@_vline_ii:
6453 {
6454     \tl_clear:N \l_@@_tikz_rule_tl
6455     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6456     \bool_if:NTF \l_@@_dotted_bool
6457     { \@@_vline_iv: }
6458     {
6459         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6460         { \@@_vline_iii: }
6461         { \@@_vline_v: }
6462     }
6463 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

6464 \cs_new_protected:Npn \@@_vline_iii:
6465 {
6466     \pgfpicture
6467     \pgfrememberpicturepositiononpagetrue
6468     \pgf@relevantforpicturesizefalse
6469     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6470     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6471     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6472     \dim_set:Nn \l_tmpb_dim
6473     {
6474         \pgf@x
6475         - 0.5 \l_@@_rule_width_dim
6476         +
6477         ( \arrayrulewidth * \l_@@_multiplicity_int
6478           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6479     }
6480     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6481     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6482     \bool_lazy_all:nT
6483     {
6484         { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }

```

```

6485     { \cs_if_exist_p:N \CT@drsc@ }
6486     { ! \tl_if_blank_p:o \CT@drsc@ }
6487   }
6488   {
6489     \group_begin:
6490     \CT@drsc@
6491     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6492     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6493     \dim_set:Nn \l_@@_tmpd_dim
6494       {
6495         \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6496         * ( \l_@@_multiplicity_int - 1 )
6497       }
6498     \pgfpathrectanglecorners
6499       { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6500       { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6501     \pgfusepath { fill }
6502     \group_end:
6503   }
6504   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6505   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6506   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6507   {
6508     \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6509     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6510     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6511   }
6512   \CT@arc@
6513   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6514   \pgfsetrectcap
6515   \pgfusepathqstroke
6516   \endpgfpicture
6517 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6518 \cs_new_protected:Npn \@@_vline_iv:
6519 {
6520   \pgfpicture
6521   \pgfrememberpicturepositiononpagetrue
6522   \pgf@relevantforpicturesizefalse
6523   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6524   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6525   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6526   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6527   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6528   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6529   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6530   \CT@arc@
6531   \@@_draw_line:
6532   \endpgfpicture
6533 }

```

The following code is for the case when the user uses the key `tikz`.

```

6534 \cs_new_protected:Npn \@@_vline_v:
6535 {
6536   \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).

```

6537   \CT@arc@
6538   \tl_if_empty:NF \l_@@_rule_color_tl
6539   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }

```

```

6540 \pgfrememberpicturepositiononpagetrue
6541 \pgf@relevantforpicturesizefalse
6542 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6543 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6544 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6545 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6546 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6547 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6548 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6549 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6550 ( \l_tmpb_dim , \l_tmpa_dim ) --
6551 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6552 \end { tikzpicture }
6553 }

```

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).

```

6554 \cs_new_protected:Npn \@@_draw_vlines:
6555 {
6556   \int_step_inline:nnn
6557   {
6558     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6559     { 2 }
6560     { 1 }
6561   }
6562   {
6563     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6564     { \c@jCol }
6565     { \int_eval:n { \c@jCol + 1 } }
6566   }
6567   {
6568     \str_if_eq:eeF { \l_@@_vlines_clist } { all }
6569     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6570     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6571   }
6572 }

```

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}`.

```

6573 \cs_new_protected:Npn \@@_hline:n #1
6574 {

```

The group is for the options.

```

6575   \group_begin:
6576   \int_set_eq:NN \l_@@_end_int \c@jCol
6577   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6578   \@@_hline_i:
6579   \group_end:
6580 }

6581 \cs_new_protected:Npn \@@_hline_i:
6582 {

```

`\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`.

```

6583   \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6584   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6585   \l_tmpb_tl
6586   {

```


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.

```
6587 \bool_gset_true:N \g_tmpa_bool
```

We test whether we are in a block.

```
6588 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6589 { \@@_test_hline_in_block:nnnnn ##1 }

6590 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6591 { \@@_test_hline_in_block:nnnnn ##1 }

6592 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6593 { \@@_test_hline_in_stroken_block:nnnn ##1 }

6594 \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_h: }
6595 \bool_if:NTF \g_tmpa_bool
6596 {
6597   \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.

```
6598   { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6599 }
6600 {
6601   \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6602   {
6603     \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6604     \@@_hline_ii:
6605     \int_zero:N \l_@@_local_start_int
6606   }
6607 }
6608 }
6609 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6610 {
6611   \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6612   \@@_hline_ii:
6613 }
6614 }
```

```
6615 \cs_new_protected:Npn \@@_test_in_corner_h:
6616 {
6617   \int_compare:nNnTF { \l_tmpa_tl } = { \c_iRow + 1 }
6618   {
6619     \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6620     { \bool_set_false:N \g_tmpa_bool }
6621   }
6622   {
6623     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6624     {
6625       \int_compare:nNnTF { \l_tmpa_tl } = { \c_one_int }
6626       { \bool_set_false:N \g_tmpa_bool }
6627       {
6628         \@@_if_in_corner:nT
6629         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6630         { \bool_set_false:N \g_tmpa_bool }
6631       }
6632     }
6633   }
6634 }
```

```
6635 \cs_new_protected:Npn \@@_hline_ii:
6636 {
```

```

6637 \tl_clear:N \l_@@_tikz_rule_tl
6638 \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6639 \bool_if:NTF \l_@@_dotted_bool
6640 { \@@_hline_iv: }
6641 {
6642   \tl_if_empty:NTF \l_@@_tikz_rule_tl
6643   { \@@_hline_iii: }
6644   { \@@_hline_v: }
6645 }
6646 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6647 \cs_new_protected:Npn \@@_hline_iii:
6648 {
6649   \pgfpicture
6650   \pgfrememberpicturepositiononpagetrue
6651   \pgf@relevantforpicturesizefalse
6652   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6653   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6654   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6655   \dim_set:Nn \l_tmpb_dim
6656   {
6657     \pgf@y
6658     - 0.5 \l_@@_rule_width_dim
6659     +
6660     ( \arrayrulewidth * \l_@@_multiplicity_int
6661       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6662   }
6663   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6664   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6665   \bool_lazy_all:nT
6666   {
6667     { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6668     { \cs_if_exist_p:N \CT@drsc@ }
6669     { ! \tl_if_blank_p:o \CT@drsc@ }
6670   }
6671   {
6672     \group_begin:
6673     \CT@drsc@
6674     \dim_set:Nn \l_@@_tmpd_dim
6675     {
6676       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6677       * ( \l_@@_multiplicity_int - 1 )
6678     }
6679     \pgfpathrectanglecorners
6680     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6681     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6682     \pgfusepathqfill
6683     \group_end:
6684   }
6685   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6686   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6687   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6688   {
6689     \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6690     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6691     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6692   }
6693   \CT@arc@
6694   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6695   \pgfsetrectcap
6696   \pgfusepathqstroke
6697   \endpgfpicture

```

6698 }

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```
\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```
\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```
6699 \cs_new_protected:Npn \@@_hline_iv:
6700 {
6701   \pgfpicture
6702   \pgfrememberpicturepositiononpagetrue
6703   \pgf@relevantforpicturesizefalse
6704   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6705   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6706   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6707   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6708   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6709   \int_compare:nNnT { \l_@@_local_start_int } = { \c_one_int }
6710   {
6711     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6712     \bool_if:NF \g_@@_delims_bool
6713     { \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.

```
6714     \tl_if_eq:NnF \g_@@_left_delim_tl (
6715       { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6716     )
6717     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6718     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6719     \int_compare:nNnT { \l_@@_local_end_int } = { \c_jCol }
6720     {
6721       \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6722       \bool_if:NF \g_@@_delims_bool
6723       { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6724       \tl_if_eq:NnF \g_@@_right_delim_tl )
6725       { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6726     }
6727     \CT@arc@
6728     \@@_draw_line:
6729     \endpgfpicture
6730   }
```

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`).

```
6731 \cs_new_protected:Npn \@@_hline_v:
6732 {
6733   \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).

```

6734 \CT@arc@
6735 \tl_if_empty:NF \l_@@_rule_color_tl
6736 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6737 \pgfrememberpicturepositiononpagetrue
6738 \pgf@relevantforpicturesizefalse
6739 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6740 \dim_set_eq:NN \l_tmpa_dim \pgf@x
6741 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6742 \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6743 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6744 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6745 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6746 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6747 ( \l_tmpa_dim , \l_tmpb_dim ) --
6748 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6749 \end { tikzpicture }
6750 }

```

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).

```

6751 \cs_new_protected:Npn \@@_draw_hlines:
6752 {
6753   \int_step_inline:nnn
6754   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6755   {
6756     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6757     { \c@iRow }
6758     { \int_eval:n { \c@iRow + 1 } }
6759   }
6760   {
6761     \str_if_eq:eeF { \l_@@_hlines_clist } { all }
6762     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6763     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6764   }
6765 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6766 \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.

```

6767 \cs_set:Npn \@@_Hline_i:n #1
6768 {
6769   \peek_remove_spaces:n
6770   {
6771     \peek_meaning:NTF \Hline
6772     { \@@_Hline_ii:nn { #1 + 1 } }
6773     { \@@_Hline_iii:n { #1 } }
6774   }
6775 }
6776 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6777 \cs_set:Npn \@@_Hline_iii:n #1
6778 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6779 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6780 {
6781   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6782   \skip_vertical:N \l_@@_rule_width_dim
6783   \tl_gput_right:Ne \g_@@_pre_code_after_tl

```

```

6784     {
6785         \@@_hline:n
6786         {
6787             multiplicity = #1 ,
6788             position = \int_eval:n { \c@iRow + 1 } ,
6789             total-width = \dim_use:N \l_@@_rule_width_dim ,
6790             #2
6791         }
6792     }
6793 \egroup
6794 }

```

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`).

```

6795 \cs_new_protected:Npn \@@_custom_line:n #1
6796 {
6797     \str_clear_new:N \l_@@_command_str
6798     \str_clear_new:N \l_@@_ccommand_str
6799     \str_clear_new:N \l_@@_letter_str
6800     \tl_clear_new:N \l_@@_other_keys_tl
6801     \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
6802     \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
6803     {
6804         \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }

```

We delete the last character which is a space.

```

6805         \str_set:Ne \l_@@_command_str
6806         { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
6807     }
6808     \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
6809     {
6810         \str_set:Ne \l_@@_ccommand_str
6811         { \str_tail:N \l_@@_ccommand_str }
6812         \str_set:Ne \l_@@_ccommand_str
6813         { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
6814     }

```

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.

```

6815     \bool_lazy_all:nTF
6816     {
6817         { \str_if_empty_p:N \l_@@_letter_str }
6818         { \str_if_empty_p:N \l_@@_command_str }
6819         { \str_if_empty_p:N \l_@@_ccommand_str }
6820     }
6821     { \@@_error:n { No~letter~and~no~command } }
6822     { \@@_custom_line_i:o \l_@@_other_keys_tl }
6823 }
6824 \keys_define:nn { nicematrix / custom-line }
6825 {
6826     letter .str_set:N = \l_@@_letter_str ,
6827     letter .value_required:n = true ,
6828     command .str_set:N = \l_@@_command_str ,
6829     command .value_required:n = true ,
6830     ccommand .str_set:N = \l_@@_ccommand_str ,
6831     ccommand .value_required:n = true ,
6832 }

```

```

6833 \cs_new_protected:Npn \l_@@_custom_line_i:n #1
6834 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6835 \bool_set_false:N \l_@@_tikz_rule_bool
6836 \bool_set_false:N \l_@@_dotted_rule_bool
6837 \bool_set_false:N \l_@@_color_bool
6838 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6839 \bool_if:NT \l_@@_tikz_rule_bool
6840 {
6841   \IfPackageLoadedF { tikz }
6842   { \@@_error:n { tikz-in-custom-line-without-tikz } }
6843   \bool_if:NT \l_@@_color_bool
6844   { \@@_error:n { color-in-custom-line-with-tikz } }
6845 }
6846 \bool_if:NT \l_@@_dotted_rule_bool
6847 {
6848   \int_compare:nNt { \l_@@_multiplicity_int } > { \c_one_int }
6849   { \@@_error:n { key-multiplicity-with-dotted } }
6850 }
6851 \str_if_empty:NF \l_@@_letter_str
6852 {
6853   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6854   { \@@_error:n { Several-letters } }
6855   {
6856     \tl_if_in:NoTF
6857     \c_@@_forbidden_letters_str
6858     \l_@@_letter_str
6859     { \@@_error:ne { Forbidden-letter } \l_@@_letter_str }
6860   }

```

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.

```

6861 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6862 { \@@_v_custom_line:nn { #1 } }
6863 }
6864 }
6865 }
6866 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6867 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6868 }
6869 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6870 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6871 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{nicematrix/Rules}`). That's why the following set of keys has some keys which are no-op.

```

6872 \keys_define:nn { nicematrix / custom-line-bis }
6873 {
6874   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6875   multiplicity .initial:n = 1 ,
6876   multiplicity .value_required:n = true ,
6877   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6878   color .value_required:n = true ,
6879   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6880   tikz .value_required:n = true ,
6881   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6882   dotted .value_forbidden:n = true ,

```

```

6883 total-width .code:n = { } ,
6884 total-width .value_required:n = true ,
6885 width .code:n = { } ,
6886 width .value_required:n = true ,
6887 sep-color .code:n = { } ,
6888 sep-color .value_required:n = true ,
6889 unknown .code:n =
6890   \@@_unknown_key:nn
6891   { nicematrix / custom-line-bis }
6892   { Unknown-key-for-custom-line }
6893 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6894 \bool_new:N \l_@@_dotted_rule_bool
6895 \bool_new:N \l_@@_tikz_rule_bool
6896 \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`.

```

6897 \keys_define:nn { nicematrix / custom-line-width }
6898 {
6899   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6900   multiplicity .initial:n = 1 ,
6901   multiplicity .value_required:n = true ,
6902   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6903   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6904   \bool_set_true:N \l_@@_total_width_bool ,
6905   total-width .value_required:n = true ,
6906   width .meta:n = { total-width = #1 } ,
6907   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6908 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule (hence the ‘h’ in the name) with the full width of the array. `#1` is the whole set of keys to pass to the command `\@@_hline:n` (which is in the internal `\CodeAfter`).

```

6909 \cs_new_protected:Npn \@@_h_custom_line:n #1
6910 {

```

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`).

```

6911   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6912   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6913 }

```

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`).

```

6914 \cs_new_protected:Npn \@@_c_custom_line:n #1
6915 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

6916   \exp_args:Nc \NewExpandableDocumentCommand
6917   { nicematrix - \l_@@_ccommand_str }
6918   { 0 { } m }
6919   {
6920     \noalign
6921     {
6922       \@@_compute_rule_width:n { #1 , ##1 }
6923       \skip_vertical:n { \l_@@_rule_width_dim }

```

```

6924         \clist_map_inline:nn
6925         { ##2 }
6926         { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6927     }
6928 }
6929 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6930 }

```

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*.

```

6931 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6932 {
6933     \tl_if_in:nnTF { #2 } { - }
6934     { \@@_cut_on_hyphen:w #2 \q_stop }
6935     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6936     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6937     {
6938         \@@_hline:n
6939         {
6940             #1 ,
6941             start = \l_tmpa_tl ,
6942             end = \l_tmpb_tl ,
6943             position = \int_eval:n { \c@iRow + 1 } ,
6944             total-width = \dim_use:N \l_@@_rule_width_dim
6945         }
6946     }
6947 }
6948 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6949 {
6950     \bool_set_false:N \l_@@_tikz_rule_bool
6951     \bool_set_false:N \l_@@_total_width_bool
6952     \bool_set_false:N \l_@@_dotted_rule_bool
6953     \keys_set:nn { nicematrix / custom-line-width } { #1 }
6954     \bool_if:NF \l_@@_total_width_bool
6955     {
6956         \bool_if:NTF \l_@@_dotted_rule_bool
6957         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6958         {
6959             \bool_if:NF \l_@@_tikz_rule_bool
6960             {
6961                 \dim_set:Nn \l_@@_rule_width_dim
6962                 {
6963                     \arrayrulewidth * \l_@@_multiplicity_int
6964                     + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6965                 }
6966             }
6967         }
6968     }
6969 }

```

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

```

6970 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
6971 {
6972     \str_if_eq:nnTF { #2 } { [ ]
6973     { \@@_v_custom_line_i:nw { #1 } [ ] }
6974     { \@@_v_custom_line_ii:nn { #2 } { #1 } }
6975 }
6976 \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
6977 { \@@_v_custom_line:nn { #1 , #2 } }
6978 \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
6979 {
6980     \@@_compute_rule_width:n { #2 }

```


In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6981 \tl_gput_right:Ne \g_@@_array_preamble_tl
6982 { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6983 \tl_gput_right:Ne \g_@@_pre_code_after_tl
6984 {
6985   \@@_vline:n
6986   {
6987     #2 ,
6988     position = \int_eval:n { \c@jCol + 1 } ,
6989     total-width = \dim_use:N \l_@@_rule_width_dim
6990   }
6991 }
6992 \@@_rec_preamble:n #1
6993 }
6994 \@@_custom_line:n
6995 { letter = : , command = hdottedline , ccommand = cdottedline, dotted }

```

The key hvlines

The following command tests whether the current position in the array (given by `\l_tmpa_tl` for the row and `\l_tmpb_tl` for the column) would provide an horizontal rule towards the right in the block delimited by the four arguments #1, #2, #3 and #4. If this rule would be in the block (it must not be drawn), the boolean `\l_tmpa_bool` is set to false.

```

6996 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6997 {
6998   \int_compare:nNnT { \l_tmpa_tl } > { #1 }
6999   {
7000     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
7001     {
7002       \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
7003       {
7004         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
7005         { \bool_gset_false:N \g_tmpa_bool }
7006       }
7007     }
7008   }
7009 }

```

The same for vertical rules.

```

7010 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
7011 {
7012   \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
7013   {
7014     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
7015     {
7016       \int_compare:nNnT { \l_tmpb_tl } > { #2 }
7017       {
7018         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
7019         { \bool_gset_false:N \g_tmpa_bool }
7020       }
7021     }
7022   }
7023 }
7024 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
7025 {
7026   \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
7027   {
7028     \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
7029     {
7030       \int_compare:nNnTF { \l_tmpa_tl } = { #1 }
7031       { \bool_gset_false:N \g_tmpa_bool }
7032     }

```

```

7033         \int_compare:nNnT { \l_tmpa_tl } = { #3 + 1 }
7034         { \bool_gset_false:N \g_tmpa_bool }
7035     }
7036 }
7037 }
7038 }
7039 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
7040 {
7041     \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
7042     {
7043         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
7044         {
7045             \int_compare:nNnTF { \l_tmpb_tl } = { #2 }
7046             { \bool_gset_false:N \g_tmpa_bool }
7047             {
7048                 \int_compare:nNnT { \l_tmpb_tl } = { #4 + 1 }
7049                 { \bool_gset_false:N \g_tmpa_bool }
7050             }
7051         }
7052     }
7053 }

```

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.

```

7054 \cs_new_protected:Npn \@@_compute_corners:
7055 {
7056     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7057     { \@@_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`).

```

7058     \clist_clear:N \l_@@_corners_cells_clist
7059     \clist_map_inline:Nn \l_@@_corners_clist
7060     {
7061         \str_case:nnF { ##1 }
7062         {
7063             { NW }
7064             { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
7065             { NE }
7066             { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
7067             { SW }
7068             { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
7069             { SE }
7070             { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
7071         }
7072     { \@@_error:nn { bad~corner } { ##1 } }
7073 }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

7074     \clist_if_empty:NF \l_@@_corners_cells_clist
7075     {

```

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.

```

7076     \tl_gput_right:Ne \g_@@_aux_tl
7077     {
7078         \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
7079         { \l_@@_corners_cells_clist }
7080     }
7081 }
7082 }

7083 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
7084 {
7085     \int_step_inline:nnn { #1 } { #3 }
7086     {
7087         \int_step_inline:nnn { #2 } { #4 }
7088         { \cs_set_nopar:cpn { @@ _ block _ ##1 - ####1 } { } }
7089     }
7090 }

7091 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
7092 {
7093     \cs_if_exist:cTF
7094     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
7095     { \prg_return_true: }
7096     { \prg_return_false: }
7097 }

```

“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.

```

7098 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
7099 {

```

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.

```

7100     \bool_set_false:N \l_tmpa_bool
7101     \int_zero_new:N \l_@@_last_empty_row_int
7102     \int_set:Nn \l_@@_last_empty_row_int { #1 }
7103     \int_step_inline:nnnn { #1 } { #3 } { #5 }
7104     {
7105         \bool_lazy_or:nnTF
7106         {
7107             \cs_if_exist_p:c
7108             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
7109         }
7110         { \@@_if_in_block_p:nn { ##1 } { #2 } }
7111         { \bool_set_true:N \l_tmpa_bool }
7112     }
7113     \bool_if:NF \l_tmpa_bool
7114     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
7115 }
7116 }

```

Now, you determine the last empty cell in the row of number 1.

```

7117 \bool_set_false:N \l_tmpa_bool
7118 \int_zero_new:N \l_@@_last_empty_column_int
7119 \int_set:Nn \l_@@_last_empty_column_int { #2 }
7120 \int_step_inline:nnnn { #2 } { #4 } { #6 }
7121 {
7122   \bool_lazy_or:nnTF
7123   {
7124     \cs_if_exist_p:c
7125     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
7126   }
7127   { \@@_if_in_block_p:nn { #1 } { ##1 } }
7128   { \bool_set_true:N \l_tmpa_bool }
7129   {
7130     \bool_if:NF \l_tmpa_bool
7131     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
7132   }
7133 }

```

Now, we loop over the rows.

```

7134 \int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int }
7135 {

```

We treat the row number ##1 with another loop.

```

7136 \bool_set_false:N \l_tmpa_bool
7137 \int_step_inline:nnnn { #2 } { #4 } { \l_@@_last_empty_column_int }
7138 {
7139   \bool_lazy_or:nnTF
7140   { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7141   { \@@_if_in_block_p:nn { ##1 } { #####1 } }
7142   { \bool_set_true:N \l_tmpa_bool }
7143   {
7144     \bool_if:NF \l_tmpa_bool
7145     {
7146       \int_set:Nn \l_@@_last_empty_column_int { #####1 }
7147       \clist_put_right:Nn
7148       \l_@@_corners_cells_clist
7149       { ##1 - #####1 }
7150       \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7151     }
7152   }
7153 }
7154 }
7155 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

7156 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
7157 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }

```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient: `\clist_if_in:NnT \l_@@_corners_cells_clist { #1 } ...`

24 The environment {NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```

7158 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

7159 \keys_define:nn { nicematrix / NiceMatrixBlock }
7160 {
7161   auto-columns-width .code:n =
7162   {
7163     \bool_set_true:N \l_@@_block_auto_columns_width_bool
7164     \dim_gzero_new:N \g_@@_max_cell_width_dim
7165     \bool_set_true:N \l_@@_auto_columns_width_bool
7166   }
7167 }

7168 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
7169 {
7170   \int_gincr:N \g_@@_NiceMatrixBlock_int
7171   \dim_zero:N \l_@@_columns_width_dim
7172   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
7173   \bool_if:NT \l_@@_block_auto_columns_width_bool
7174   {
7175     \cs_if_exist:cT
7176     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
7177     {
7178       \dim_set:Nn \l_@@_columns_width_dim
7179       {
7180         \use:c
7181         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7182       }
7183     }
7184   }
7185 }

```

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`).

```

7186 {
7187   \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.

```

7188   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
7189   {
7190     \bool_if:NT \l_@@_block_auto_columns_width_bool
7191     {
7192       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
7193       \iow_shipout:Ne \@mainaux
7194       {
7195         \cs_gset:cpn
7196         { @@ _ 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.

```

7197         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7198       }
7199       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
7200     }
7201   }
7202   \ignorespacesafterend
7203 }

```

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).

```

7204 \cs_new_protected:Npn \@@_create_extra_nodes:
7205 {
7206   \bool_if:nTF \l_@@_medium_nodes_bool
7207   {
7208     \bool_if:NTF \l_@@_no_cell_nodes_bool
7209     { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7210     {
7211       \bool_if:NTF \l_@@_large_nodes_bool
7212       \@@_create_medium_and_large_nodes:
7213       \@@_create_medium_nodes:
7214     }
7215   }
7216   {
7217     \bool_if:NT \l_@@_large_nodes_bool
7218     {
7219       \bool_if:NTF \l_@@_no_cell_nodes_bool
7220       { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7221       \@@_create_large_nodes:
7222     }
7223   }
7224 }

```

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`.

```

7225 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7226 {
7227   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7228   {
7229     \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7230     \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7231     \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7232     \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7233   }
7234   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7235   {
7236     \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7237     \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7238     \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7239     \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7240   }

```

We begin the two nested loops over the rows and the columns of the array.

```

7241 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7242 {
7243   \int_step_variable:nnNn
7244   \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.

```

7245 {
7246   \cs_if_exist:cT
7247   { 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`.

```

7248 {
7249   \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7250   \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7251   { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7252   \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7253   {
7254     \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7255     { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7256   }

```

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`.

```

7257   \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7258   \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7259   { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } { \pgf@y } }
7260   \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7261   {
7262     \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7263     { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } { \pgf@x } }
7264   }
7265 }
7266 }
7267 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7268 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7269 {
7270   \dim_compare:nNnT
7271   { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7272   {
7273     \@@_qpoint:n { row - \@@_i: - base }
7274     \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7275     \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7276   }
7277 }
7278 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7279 {
7280   \dim_compare:nNnT
7281   { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7282   {
7283     \@@_qpoint:n { col - \@@_j: }
7284     \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7285     \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7286   }
7287 }
7288 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7289 \cs_new_protected:Npn \@@_create_medium_nodes:
7290 {
7291   \pgfpicture
7292   \pgfrememberpicturepositiononpagetrue
7293   \pgf@relevantforpicturesizefalse
7294   \@@_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”.

```

7295   \tl_set:Nn \l_@@_suffix_tl { -medium }
7296   \@@_create_nodes:
7297   \endpgfpicture
7298 }

```

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:`.

```

7299 \cs_new_protected:Npn \@@_create_large_nodes:
7300 {
7301   \pgfpicture
7302   \pgfrememberpicturepositiononpagetrue
7303   \pgf@relevantforpicturesizefalse
7304   \@@_computations_for_medium_nodes:
7305   \@@_computations_for_large_nodes:
7306   \tl_set:Nn \l_@@_suffix_tl { - large }
7307   \@@_create_nodes:
7308   \endpgfpicture
7309 }
7310 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7311 {
7312   \pgfpicture
7313   \pgfrememberpicturepositiononpagetrue
7314   \pgf@relevantforpicturesizefalse
7315   \@@_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”.

```

7316   \tl_set:Nn \l_@@_suffix_tl { - medium }
7317   \@@_create_nodes:
7318   \@@_computations_for_large_nodes:
7319   \tl_set:Nn \l_@@_suffix_tl { - large }
7320   \@@_create_nodes:
7321   \endpgfpicture
7322 }

```

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.

```

7323 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7324 {
7325   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7326   \int_set_eq:NN \l_@@_first_col_int \c_one_int

```

We have to change the values of all the dimensions `l_@@_row_i_min_dim`, `l_@@_row_i_max_dim`, `l_@@_column_j_min_dim` and `l_@@_column_j_max_dim`.

```

7327   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7328   {
7329     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }

```

¹⁵If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`


```

7330     {
7331     (
7332         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7333         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7334     )
7335     / 2
7336     }
7337     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7338     { l_@@_row _ \@@_i: _ min_dim }
7339 }
7340 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7341 {
7342     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7343     {
7344     (
7345         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7346         \dim_use:c
7347         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7348     )
7349     / 2
7350     }
7351     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7352     { l_@@_column _ \@@_j: _ max _ dim }
7353 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7354     \dim_sub:cn
7355     { l_@@_column _ 1 _ min _ dim }
7356     \l_@@_left_margin_dim
7357     \dim_add:cn
7358     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7359     \l_@@_right_margin_dim
7360 }

```

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).

```

7361 \cs_new_protected:Npn \@@_create_nodes:
7362 {
7363     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7364     {
7365         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7366         {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7367         \@@_pgf_rect_node:nnnnn
7368         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7369         { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7370         { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7371         { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7372         { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7373         \str_if_empty:NF \l_@@_name_str
7374         {
7375             \pgfnodealias
7376             { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7377             { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7378         }
7379     }
7380 }
7381 \int_step_inline:nn { \c@iRow }

```

```

7382     {
7383       \pgfnodealias
7384       { \@@_env: - ##1 - last \l_@@_suffix_tl }
7385       { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7386     }
7387   \int_step_inline:nn { \c@jCol }
7388   {
7389     \pgfnodealias
7390     { \@@_env: - last - ##1 \l_@@_suffix_tl }
7391     { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7392   }
7393   \pgfnodealias % added 2025-04-05
7394   { \@@_env: - last - last \l_@@_suffix_tl }
7395   { \@@_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 .

```

7396     \seq_map_pairwise_function:NNN
7397     \g_@@_multicolumn_cells_seq
7398     \g_@@_multicolumn_sizes_seq
7399     \@@_node_for_multicolumn:nn
7400   }

7401 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7402 {
7403   \cs_set_nopar:Npn \@@_i: { #1 }
7404   \cs_set_nopar:Npn \@@_j: { #2 }
7405 }

```

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”).

```

7406 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7407 {
7408   \@@_extract_coords_values: #1 \q_stop
7409   \@@_pgf_rect_node:nnnnn
7410   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7411   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7412   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7413   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7414   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7415   \str_if_empty:NF \l_@@_name_str
7416   {
7417     \pgfnodealias
7418     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7419     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7420   }
7421 }

```

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).

```

7422 \keys_define:nn { nicematrix / Block / FirstPass }
7423 {
7424   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7425     \bool_set_true:N \l_@@_p_block_bool ,
7426   j .value_forbidden:n = true ,
7427   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7428   l .value_forbidden:n = true ,
7429   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7430   r .value_forbidden:n = true ,
7431   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7432   c .value_forbidden:n = true ,
7433   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7434   L .value_forbidden:n = true ,
7435   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7436   R .value_forbidden:n = true ,
7437   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7438   C .value_forbidden:n = true ,
7439   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7440   t .value_forbidden:n = true ,
7441   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7442   T .value_forbidden:n = true ,
7443   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7444   b .value_forbidden:n = true ,
7445   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7446   B .value_forbidden:n = true ,
7447   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7448   m .value_forbidden:n = true ,
7449   v-center .meta:n = m ,
7450   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7451   p .value_forbidden:n = true ,
7452   color .code:n =
7453     \@@_color:n { #1 }
7454     \tl_set_rescan:Nnn
7455       \l_@@_draw_tl
7456       { \char_set_catcode_other:N ! }
7457       { #1 } ,
7458   color .value_required:n = true ,
7459   respect-arraystretch .code:n =
7460     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7461   respect-arraystretch .value_forbidden:n = true ,
7462 }

```

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.

```

7463 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7464 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7465 {

```

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).

```

7466   \tl_if_blank:nTF { #2 }
7467   { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7468   {
7469     \tl_if_in:nnTF { #2 } { - }
7470     {
7471       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7472       \@@_Block_i_czech:w \@@_Block_i:w
7473       #2 \q_stop
7474     }
7475     {
7476       \@@_error:nn { Bad~argument~for~Block } { #2 }

```

```

7477         \@@_Block_ii:nnnnn \c_one_int \c_one_int
7478     }
7479 }
7480 { #1 } { #3 } { #4 }
7481 \ignorespaces
7482 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7483 \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`.

```

7484 {
7485   \char_set_catcode_active:N -
7486   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7487 }

```

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.

```

7488 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7489 {

```

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).

```

7490   \bool_lazy_or:nnTF
7491     { \tl_if_blank_p:n { #1 } }
7492     { \str_if_eq_p:ee { * } { #1 } }
7493     { \int_set:Nn \l_tmpa_int { 100 } }
7494     { \int_set:Nn \l_tmpa_int { #1 } }
7495   \bool_lazy_or:nnTF
7496     { \tl_if_blank_p:n { #2 } }
7497     { \str_if_eq_p:ee { * } { #2 } }
7498     { \int_set:Nn \l_tmpb_int { 100 } }
7499     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7500   \int_compare:nNnTF { \l_tmpb_int } = { \c_one_int }
7501   {
7502     \tl_if_empty:NTF \l_@@_hpos_cell_tl
7503       { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7504       { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7505   }
7506   { \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.

```

7507   \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7508   \tl_set:Ne \l_tmpa_tl
7509   {
7510     { \int_use:N \c@iRow }
7511     { \int_use:N \c@jCol }
7512     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7513     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7514   }

```

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).

```
7515 \bool_set_false:N \l_tmpa_bool
7516 \bool_if:NT \l_@@_amp_in_blocks_bool
```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```
7517 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7518 \bool_case:nF
7519 {
7520 \l_tmpa_bool { \@@_Block_vii:eenenn }
7521 \l_@@_p_block_bool { \@@_Block_vi:eenenn }
```

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.

```
7522 \l_@@_X_bool { \@@_Block_v:eenenn }
7523 { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eenenn }
7524 { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eenenn }
7525 { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eenenn }
7526 }
7527 { \@@_Block_v:eenenn }
7528 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7529 }
```

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.

```
7530 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7531 {
7532 \int_gincr:N \g_@@_block_box_int
7533 \cs_set_eq:NN \cellcolor \@@_cellcolor_error
7534 \cs_set_eq:NN \rowcolor \@@_rowcolor_error
7535 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7536 {
7537 \tl_gput_right:Ne \g_@@_pre_code_after_tl
7538 {
7539 \@@_actually_diagbox:nnnnnn
7540 { \int_use:N \c@iRow }
7541 { \int_use:N \c@jCol }
7542 { \int_eval:n { \c@iRow + #1 - 1 } }
7543 { \int_eval:n { \c@jCol + #2 - 1 } }
7544 { \g_@@_row_style_tl \exp_not:n { ##1 } }
7545 { \g_@@_row_style_tl \exp_not:n { ##2 } }
7546 }
7547 }
7548 \box_gclear_new:c
7549 { 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!

```

7550   \hbox_gset:cn
7551   { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7552   {

```

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`).

```

7553   \tl_if_empty:NTF \l_@@_color_tl
7554   { \int_compare:nNnT { #2 } = { \c_one_int } { \set@color } }
7555   { \@@_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`.

```

7556   \int_compare:nNnT { #1 } = { \c_one_int }
7557   {
7558       \int_if_zero:nTF { \c@iRow }
7559       {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & & -4 & 5 & & \\
3 & & -4 & 5 & & -6 & \\
-4 & 5 & & -6 & 7 & & \\
5 & & -6 & 7 & & -8 & \\
\end{bNiceMatrix}$

```

```

7560   \cs_set_eq:NN \Block \@@_NullBlock:
7561   \l_@@_code_for_first_row_tl
7562   }
7563   {
7564       \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7565       {
7566           \cs_set_eq:NN \Block \@@_NullBlock:
7567           \l_@@_code_for_last_row_tl
7568       }
7569   }
7570   \g_@@_row_style_tl
7571   }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7572   \@@_reset_arraystretch:
7573   \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7574   #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.).

```
7575 \@@_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}`.

```
7576 \bool_if:NTF \l_@@_tabular_bool
7577 {
7578   \bool_lazy_all:nTF
7579   {
7580     { \int_compare_p:nNn { #2 } = { \c_one_int } }
```

Remind that, when the column has not a fixed width, the dimension `\l_@@_col_width_dim` has the conventional value of -1 cm.

```
7581   {
7582     ! \dim_compare_p:nNn
7583     { \l_@@_col_width_dim } < { \c_zero_dim }
7584   }
7585   { ! \g_@@_rotate_bool }
7586 }
```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```
7587 {
7588   \use:e
7589   {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
7590 \exp_not:N \begin { minipage }
7591 [ \str_lowercase:f \l_@@_vpos_block_str ]
7592 { \l_@@_col_width_dim }
7593 \str_case:on \l_@@_hpos_block_str
7594 { c \centering r \raggedleft l \raggedright }
7595 }
7596 #5
7597 \end { minipage }
7598 }
```

In the other cases, we use a `{tabular}`.

```
7599 {
7600   \use:e
7601   {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
7602 \exp_not:N \begin { tabular }
7603 [ \str_lowercase:f \l_@@_vpos_block_str ]
7604 { @ { } \l_@@_hpos_block_str @ { } }
7605 }
7606 #5
7607 \end { tabular }
7608 }
7609 }
```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```
7610 {
7611   $ % $
7612   \use:e
7613   {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7614         \exp_not:N \begin { array }
7615         [ \str_lowercase:f \l_@@_vpos_block_str ]
7616         { @ { } \l_@@_hpos_block_str @ { } }
7617     }
7618     #5
7619     \end { array }
7620     $ % $
7621 }
7622 }
```

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).

```

7623     \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.

```

7624     \int_compare:nNnT { #2 } = { \c_one_int }
7625     {
7626         \dim_gset:Nn \g_@@_blocks_wd_dim
7627         {
7628             \dim_max:nn
7629             { \g_@@_blocks_wd_dim }
7630             {
7631                 \box_wd:c
7632                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7633             }
7634         }
7635     }
```

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.

```

7636     \int_compare:nNnT { #1 } = { \c_one_int }
7637     {
7638         \bool_lazy_any:nT
7639         {
7640             { \str_if_empty_p:N \l_@@_vpos_block_str }
7641             { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
7642             { \str_if_eq_p:ee { \l_@@_vpos_block_str } { b } }
7643         }
7644         { \@@_adjust_blocks_ht_dp: }
7645     }
7646     \seq_gput_right:Ne \g_@@_blocks_seq
7647     {
7648         \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.

```

7649     {
7650         \exp_not:n { #3 } ,
7651         \l_@@_hpos_block_str ,
```

Now, we put a key for the vertical alignment.

```

7652     \bool_if:NT \g_@@_rotate_bool
7653     {
7654         \bool_if:NTF \g_@@_rotate_c_bool
7655         { m }
7656         {
```



```

7657         \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7658         { T }
7659     }
7660 }
7661 }
7662 {
7663     \box_use_drop:c
7664     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7665 }
7666 }
7667 \bool_set_false:N \g_@@_rotate_c_bool
7668 }
7669 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7670 {
7671     \dim_gset:Nn \g_@@_blocks_ht_dim
7672     {
7673         \dim_max:nn
7674         { \g_@@_blocks_ht_dim }
7675         {
7676             \box_ht:c
7677             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7678         }
7679     }
7680     \dim_gset:Nn \g_@@_blocks_dp_dim
7681     {
7682         \dim_max:nn
7683         { \g_@@_blocks_dp_dim }
7684         {
7685             \box_dp:c
7686             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7687         }
7688     }
7689 }
7690 \cs_new:Npn \@@_adjust_hpos_rotate:
7691 {
7692     \bool_if:NT \g_@@_rotate_bool
7693     {
7694         \str_set:Ne \l_@@_hpos_block_str
7695         {
7696             \bool_if:NTF \g_@@_rotate_c_bool
7697             { c }
7698             {
7699                 \str_case:onF \l_@@_vpos_block_str
7700                 { b l B l t r T r }
7701                 {
7702                     \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
7703                     { r }
7704                     { l }
7705                 }
7706             }
7707         }
7708     }
7709 }
7710 \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*.

```

7711 \cs_new_protected:Npn \@@_rotate_box_of_block:
7712 {
7713     \box_grotate:cn

```

```

7714 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7715 { 90 }
7716 \int_compare:nNt { \c@iRow } = { \l_@@_last_row_int }
7717 {
7718   \vbox_gset_top:cn
7719   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7720   {
7721     \skip_vertical:n { 0.8 ex }
7722     \box_use:c
7723     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7724   }
7725 }
7726 \bool_if:NT \g_@@_rotate_c_bool
7727 {
7728   \hbox_gset:cn
7729   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7730   {
7731     $ % $
7732     \vcenter
7733     {
7734       \box_use:c
7735       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7736     }
7737     $ % $
7738   }
7739 }
7740 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key p). In that case, the content of the block is *not* composed right away in a box. The composition in a box will be done further, just after the construction of the array (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnn`).

#1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7741 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7742 {
7743   \seq_gput_right:Ne \g_@@_blocks_seq
7744   {
7745     \l_tmpa_tl
7746     { \exp_not:n { #3 } }
7747     {
7748       \bool_if:NTF \l_@@_tabular_bool
7749       {
7750         \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7751 \@@_reset_arraystretch:
7752 \exp_not:n
7753 {
7754   \dim_zero:N \extrarowheight
7755   #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).

```

7756 \tag_if_active:T { \tag_stop:n { table } }
7757 \use:e
7758 {
7759   \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7760   { @ { } \l_@@_hpos_block_str @ { } }

```

```

7761         }
7762         #5
7763     \end { tabular }
7764 }
7765 \group_end:
7766 }

```

When we are *not* in an environment {NiceTabular} (or similar).

```

7767 {
7768     \group_begin:

```

The following will be no-op when respect-arraystretch is in force.

```

7769     \@@_reset_arraystretch:
7770     \exp_not:n
7771     {
7772         \dim_zero:N \extrarowheight
7773         #4
7774         $ % $
7775         \use:e
7776         {
7777             \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7778             { @ { } \l_@@_hpos_block_str @ { } }
7779         }
7780         #5
7781         \end { array }
7782         $ % $
7783     }
7784     \group_end:
7785 }
7786 }
7787 }
7788 }
7789 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a \Block which uses the key p.

```

7790 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7791 {
7792     \seq_gput_right:Ne \g_@@_blocks_seq
7793     {
7794         \l_tmpa_tl
7795         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7796         { { \exp_not:n { #4 #5 } } }
7797     }
7798 }
7799 \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.

```

7800 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7801 {
7802     \seq_gput_right:Ne \g_@@_blocks_seq
7803     {
7804         \l_tmpa_tl
7805         { \exp_not:n { #3 } }
7806         { \exp_not:n { #4 #5 } }
7807     }
7808 }
7809 \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).

```

7810 \keys_define:nn { nicematrix / Block / SecondPass }
7811 {
7812     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7813     ampersand-in-blocks .default:n = true ,
7814     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7815     tikz .code:n =
7816         \IfPackageLoadedTF { tikz }
7817         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7818         { \@@_error:n { tikz~key~without~tikz } } ,
7819     tikz .value_required:n = true ,
7820     fill .code:n =
7821         \tl_set_rescan:Nnn
7822         \l_@@_fill_tl
7823         { \char_set_catcode_other:N ! }
7824         { #1 } ,
7825     fill .value_required:n = true ,

```

In fine, the opacity will be applied by `\pgfsetfillopacity`.

```

7826     opacity .tl_set:N = \l_@@_opacity_tl ,
7827     opacity .value_required:n = true ,
7828     draw .code:n =
7829         \tl_set_rescan:Nnn
7830         \l_@@_draw_tl
7831         { \char_set_catcode_other:N ! }
7832         { #1 } ,
7833     draw .default:n = default ,
7834     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7835     rounded-corners .default:n = 4 pt ,
7836     color .code:n =
7837         \@@_color:n { #1 }
7838         \tl_set_rescan:Nnn
7839         \l_@@_draw_tl
7840         { \char_set_catcode_other:N ! }
7841         { #1 } ,
7842     borders .clist_set:N = \l_@@_borders_clist ,
7843     borders .value_required:n = true ,
7844     hvlines .meta:n = { vlines , hlines } ,
7845     vlines .bool_set:N = \l_@@_vlines_block_bool ,
7846     vlines .default:n = true ,
7847     hlines .bool_set:N = \l_@@_hlines_block_bool ,
7848     hlines .default:n = true ,
7849     line-width .dim_set:N = \l_@@_line_width_dim ,
7850     line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7851     j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7852             \bool_set_true:N \l_@@_p_block_bool ,
7853     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7854     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7855     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7856     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7857             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7858     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7859             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7860     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7861             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7862     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7863     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7864     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7865     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7866     m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7867     m .value_forbidden:n = true ,
7868     v-center .meta:n = m ,

```

```

7869 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7870 p .value_forbidden:n = true ,
7871 name .tl_set:N = \l_@@_block_name_str , % .str_set:N ?
7872 name .value_required:n = true ,
7873 name .initial:n = ,
7874 respect-arraystretch .code:n =
7875   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7876 respect-arraystretch .value_forbidden:n = true ,
7877 transparent .bool_set:N = \l_@@_transparent_bool ,
7878 transparent .default:n = true ,
7879 transparent .initial:n = false ,
7880 unknown .code:n =
7881   \@@_unknown_key:nn
7882     { nicematrix / Block / SecondPass }
7883     { Unknown-key-for-Block }
7884 }

```

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.

```

7885 \cs_new_protected:Npn \@@_draw_blocks:
7886 {
7887   \bool_if:NTF \c_@@_revtex_bool
7888     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7889     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7890   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7891 }
7892 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7893 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7894   \int_zero:N \l_@@_last_row_int
7895   \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”).

```

7896   \int_compare:nNnTF { #3 } > { 98 }
7897     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7898     { \int_set:Nn \l_@@_last_row_int { #3 } }
7899   \int_compare:nNnTF { #4 } > { 98 }
7900     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7901     { \int_set:Nn \l_@@_last_col_int { #4 } }
7902   \int_compare:nNnTF { \l_@@_last_col_int } > { \g_@@_col_total_int }
7903     {
7904       \bool_lazy_and:nnTF
7905         { \l_@@_preamble_bool }
7906         {
7907           \int_compare_p:n
7908             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7909         }
7910     {
7911       \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7912       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7913       \@@_msg_redirect_name:nn { columns-not-used } { none }
7914     }
7915     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7916 }

```

```

7917 {
7918   \int_compare:nNnTF { \l_@@_last_row_int } > { \g_@@_row_total_int }
7919   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7920   {
7921     \@@_Block_v:nneenn
7922     { #1 }
7923     { #2 }
7924     { \int_use:N \l_@@_last_row_int }
7925     { \int_use:N \l_@@_last_col_int }
7926     { #5 }
7927     { #6 }
7928   }
7929 }
7930 }

```

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.

```

7931 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7932 {

```

The group is for the keys.

```

7933   \group_begin:
7934   \int_compare:nNnT { #1 } = { #3 }
7935   { \str_set:Nn \l_@@_vpos_block_str { t } }
7936   \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`.

```

7937   \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7938   \bool_lazy_and:nnT
7939   { \l_@@_vlines_block_bool }
7940   { ! \l_@@_ampersand_bool }
7941   {
7942     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7943     {
7944       \@@_vlines_block:nnn
7945       { \exp_not:n { #5 } }
7946       { #1 - #2 }
7947       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7948     }
7949   }
7950   \bool_if:NT \l_@@_hlines_block_bool
7951   {
7952     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7953     {
7954       \@@_hlines_block:nnn
7955       { \exp_not:n { #5 } }
7956       { #1 - #2 }
7957       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7958     }
7959   }
7960   \bool_if:NF \l_@@_transparent_bool
7961   {
7962     \bool_lazy_and:nnF { \l_@@_vlines_block_bool } { \l_@@_hlines_block_bool }
7963     {

```

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).

```

7964     \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7965     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7966   }
7967 }

```

```

7968 \tl_if_empty:NF \l_@@_draw_tl
7969 {
7970   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7971   { \@@_error:n { hlines~with~color } }
7972   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7973   {
7974     \@@_stroke_block:nnn
7975     { \exp_not:n { #5 } }
7976     { #1 - #2 }
7977     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7978   }
7979   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7980   { { #1 } { #2 } { #3 } { #4 } }
7981 }
7982 \clist_if_empty:NF \l_@@_borders_clist
7983 {
7984   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7985   {
7986     \@@_stroke_borders_block:nnn
7987     { \exp_not:n { #5 } }
7988     { #1 - #2 }
7989     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7990   }
7991 }
7992 \tl_if_empty:NF \l_@@_fill_tl
7993 {
7994   \@@_add_opacity_to_fill:
7995   \tl_gput_right:Ne \g_@@_pre_code_before_tl
7996   {
7997     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7998     { #1 - #2 }
7999     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
8000     { \dim_use:N \l_@@_rounded_corners_dim }
8001   }
8002 }
8003 \seq_if_empty:NF \l_@@_tikz_seq
8004 {
8005   \tl_gput_right:Ne \g_nicematrix_code_before_tl
8006   {
8007     \@@_block_tikz:nnnnn
8008     { \seq_use:Nn \l_@@_tikz_seq { , } }
8009     { #1 }
8010     { #2 }
8011     { \int_use:N \l_@@_last_row_int }
8012     { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of TikZ keys.

```

8013   }
8014 }
8015 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
8016 {
8017   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8018   {
8019     \@@_actually_diagbox:nnnnnn
8020     { #1 }
8021     { #2 }
8022     { \int_use:N \l_@@_last_row_int }
8023     { \int_use:N \l_@@_last_col_int }
8024     { \exp_not:n { ##1 } }
8025     { \exp_not:n { ##2 } }

```

```

8026     }
8027 }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} &      & one    & \\
                        &      & two    & \\
three                  & four & five   & \\
six                    & seven & eight  & \\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block		one
		two
three	four	five
six	seven	eight

We highlight the node `1-1-block-short`

our block		one
		two
three	four	five
six	seven	eight

The construction of the node corresponding to the merged cells.

```

8028 \pgfpicture
8029 \pgfrememberpicturepositiononpagetrue
8030 \pgf@relevantforpicturesizefalse
8031 \@@_qpoint:n { row - #1 }
8032 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8033 \@@_qpoint:n { col - #2 }
8034 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8035 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
8036 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8037 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8038 \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.

```

8039 \@@_pgf_rect_node:nnnnn
8040 { \@@_env: - #1 - #2 - block }
8041 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8042 \str_if_empty:NF \l_@@_block_name_str
8043 {
8044   \pgfnodealias
8045   { \@@_env: - \l_@@_block_name_str }
8046   { \@@_env: - #1 - #2 - block }
8047   \str_if_empty:NF \l_@@_name_str
8048   {
8049     \pgfnodealias
8050     { \l_@@_name_str - \l_@@_block_name_str }
8051     { \@@_env: - #1 - #2 - block }
8052   }
8053 }

```

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.

```

8054 \bool_if:NF \l_@@_hpos_of_block_cap_bool
8055 {
8056   \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.

```

8057     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8058     {

```

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.

```

8059         \cs_if_exist:cT
8060         { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8061         {
8062             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8063             {
8064                 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
8065                 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
8066             }
8067         }
8068     }

```

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.

```

8069     \dim_compare:nNnT { \l_tmpb_dim } = { \c_max_dim }
8070     {
8071         \@@_qpoint:n { col - #2 }
8072         \dim_set_eq:NN \l_tmpb_dim \pgf@x
8073     }
8074     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
8075     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8076     {
8077         \cs_if_exist:cT
8078         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8079         {
8080             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8081             {
8082                 \pgfpointanchor
8083                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8084                 { east }
8085                 \dim_set:Nn \l_@@_tmpd_dim
8086                 { \dim_max:nn { \l_@@_tmpd_dim } { \pgf@x } }
8087             }
8088         }
8089     }
8090     \dim_compare:nNnT { \l_@@_tmpd_dim } = { - \c_max_dim }
8091     {
8092         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8093         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8094     }
8095     \@@_pgf_rect_node:nnnnn
8096     { \@@_env: - #1 - #2 - block - short }
8097     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8098 }

```

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.

```

8099     \bool_if:NT \l_@@_medium_nodes_bool
8100     {
8101         \@@_pgf_rect_node:nnn
8102         { \@@_env: - #1 - #2 - block - medium }
8103         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
8104         {
8105             \pgfpointanchor
8106             { \@@_env:
8107               - \int_use:N \l_@@_last_row_int
8108               - \int_use:N \l_@@_last_col_int - medium
8109             }

```

```

8110         { south-east }
8111     }
8112 }
8113 \endpgfpicture
8114

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand `&`.

```

8115 \bool_if:NTF \l_@@_ampersand_bool
8116 {
8117     \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
8118     \int_zero_new:N \l_@@_split_int
8119     \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.

```

8120     \int_zero_new:N \l_@@_first_row_int
8121     \int_zero_new:N \l_@@_first_col_int
8122     \int_zero_new:N \l_@@_last_row_int
8123     \int_zero_new:N \l_@@_last_col_int
8124     \int_set:Nn \l_@@_first_row_int { #1 }
8125     \int_set:Nn \l_@@_first_col_int { #2 }
8126     \int_set:Nn \l_@@_last_row_int { #3 }
8127     \int_set:Nn \l_@@_last_col_int { #4 }

8128     \pgfpicture
8129     \pgfrememberpicturerepositiononpagetrue
8130     \pgf@relevantforpicturesizefalse
8131     \@@_qpoint:n { row - #1 }
8132     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8133     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8134     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
8135     \@@_qpoint:n { col - #2 }
8136     \dim_set_eq:NN \l_tmpa_dim \pgf@x
8137     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8138     \dim_set:Nn \l_tmpb_dim
8139     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
8140     \bool_lazy_or:nnT
8141     { \l_@@_vlines_block_bool }
8142     { \str_if_eq_p:ee { \l_@@_vlines_clist } { all } }
8143     {
8144         \int_step_inline:nn { \l_@@_split_int - 1 }
8145         {
8146             \pgfpathmoveto
8147             {
8148                 \pgfpoint
8149                 { \l_tmpa_dim + ##1 \l_tmpb_dim }
8150                 \l_@@_tmpc_dim
8151             }
8152             \pgfpathlineto
8153             {
8154                 \pgfpoint
8155                 { \l_tmpa_dim + ##1 \l_tmpb_dim }
8156                 \l_@@_tmpd_dim
8157             }
8158             \CT@arc@
8159             \pgfsetlinewidth { 1.1 \arrayrulewidth }
8160             \pgfsetrectcap
8161             \pgfusepathqstroke
8162         }
8163     }
8164     \cs_set_eq:NN \cellcolor \@@_subcellcolor
8165     \int_zero_new:N \l_@@_split_i_int
8166     \str_if_eq:eeTF { \l_@@_vpos_block_str } { T }

```

```

8167 {
8168   \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
8169   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8170 }
8171 {
8172   \str_if_eq:eeTF { \l_@@_vpos_block_str } { B }
8173   {
8174     \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
8175     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8176   }
8177   {
8178     \bool_lazy_or:nnTF
8179     { \int_compare_p:nNn { #1 } = { #3 } }
8180     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
8181     {
8182       \@@_qpoint:n { row - #1 - base }
8183       \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8184     }
8185     {
8186       \str_if_eq:eeTF { \l_@@_vpos_block_str } { b }
8187       {
8188         \@@_qpoint:n { row - #3 - base }
8189         \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8190       }
8191       {
8192         \@@_qpoint:n { #1 - #2 - block }
8193         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8194       }
8195     }
8196   }
8197 }
8198 \int_step_inline:nn { \l_@@_split_int }
8199 {
8200   \group_begin:

```

The counter `\l_@@_split_i_int` is only for the command `\@@_subcellcolor`.

```

8201   \int_set:Nn \l_@@_split_i_int { ##1 }
8202   \dim_set:Nn \col@sep
8203   { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
8204   \pgftransformshift
8205   {
8206     \pgfpoint
8207     {
8208       \l_tmpa_dim + ##1 \l_tmpb_dim -
8209       \str_case:on \l_@@_hpos_block_str
8210       {
8211         l { \l_tmpb_dim + \col@sep }
8212         c { 0.5 \l_tmpb_dim }
8213         r { \col@sep }
8214       }
8215     }
8216     { \l_@@_tmpc_dim }
8217   }
8218   \pgfset { inner~sep = \c_zero_dim }
8219   \pgfnode
8220   { rectangle }
8221   {
8222     \str_if_eq:eeTF { \l_@@_vpos_block_str } { T }
8223     {
8224       \str_case:on \l_@@_hpos_block_str
8225       {
8226         l { north-west }
8227         c { north }
8228         r { north-east }

```

```

8229     }
8230   }
8231   {
8232     \str_if_eq:eeTF { \l_@@_vpos_block_str } { B }
8233     {
8234       \str_case:on \l_@@_hpos_block_str
8235       {
8236         l { south~west }
8237         c { south }
8238         r { south~east }
8239       }
8240     }
8241     {
8242       \bool_lazy_any:nTF
8243       {
8244         { \int_compare_p:nNn { #1 } = { #3 } }
8245         { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
8246         { \str_if_eq_p:ee { \l_@@_vpos_block_str } { b } }
8247       }
8248       {
8249         \str_case:on \l_@@_hpos_block_str
8250         {
8251           l { base~west }
8252           c { base }
8253           r { base~east }
8254         }
8255       }
8256       {
8257         \str_case:on \l_@@_hpos_block_str
8258         {
8259           l { west }
8260           c { center }
8261           r { east }
8262         }
8263       }
8264     }
8265   }
8266 }
8267 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8268 \group_end:
8269 }
8270 \endpgfpicture
8271 }

```

Now the case where there is no ampersand & in the content of the block.

```

8272 {
8273   \bool_if:NTF \l_@@_p_block_bool
8274   {

```

When the final user has used the key p, we have to compute the width.

```

8275 \pgfpicture
8276 \pgfrememberpicturepositiononpagetrue
8277 \pgf@relevantforpicturesizefalse
8278 \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8279 {
8280   \@@_qpoint:n { col - #2 }
8281   \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8282   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8283 }
8284 {
8285   \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8286   \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8287   \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8288 }

```

```

8289         \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8290     \endpgfpicture
8291     \hbox_set:Nn \l_@@_cell_box
8292     {
8293         \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8294             { \g_tmpb_dim }
8295         \str_case:on \l_@@_hpos_block_str
8296             { c \centering r \raggedleft l \raggedright j { } }
8297         #6
8298     \end { minipage }
8299 }
8300 }
8301 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
8302 \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.

```

8303 \pgfpicture
8304 \pgfrememberpicturepositiononpagetrue
8305 \pgf@relevantforpicturesizefalse
8306 \bool_lazy_any:nTF
8307 {
8308     { \str_if_empty_p:N \l_@@_vpos_block_str }
8309     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { c } }
8310     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { T } }
8311     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { B } }
8312 }
8313 {

```

If we are in the “first column”, we must put the block as if it was with the key `r`.

```

8314     \int_if_zero:nT { #2 } { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_r_str }

```

If we are in the “last column”, we must put the block as if it was with the key `l`.

```

8315     \bool_if:nT \g_@@_last_col_found_bool
8316     {
8317         \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
8318         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8319     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8320     \tl_set:Ne \l_tmpa_tl
8321     {
8322         \str_case:on \l_@@_vpos_block_str
8323         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8324             { } {
8325                 \str_case:on \l_@@_hpos_block_str
8326                 {
8327                     c { center }
8328                     l { west }
8329                     r { east }
8330                     j { center }
8331                 }
8332             }
8333         c {
8334             \str_case:on \l_@@_hpos_block_str
8335             {
8336                 c { center }
8337                 l { west }
8338                 r { east }
8339                 j { center }

```

```

8340         }
8341     }
8342 }
8343 T {
8344     \str_case:on \l_@@_hpos_block_str
8345     {
8346         c { north }
8347         l { north~west }
8348         r { north~east }
8349         j { north }
8350     }
8351 }
8352 }
8353 B {
8354     \str_case:on \l_@@_hpos_block_str
8355     {
8356         c { south }
8357         l { south~west }
8358         r { south~east }
8359         j { south }
8360     }
8361 }
8362 }
8363 }
8364 }
8365 \pgftransformshift
8366 {
8367     \pgfpointanchor
8368     {
8369         \@@_env: - #1 - #2 - block
8370         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8371     }
8372     { \l_tmpa_tl }
8373 }
8374 \pgfset { inner~sep = \c_zero_dim }
8375 \pgfnode
8376 { rectangle }
8377 { \l_tmpa_tl }
8378 { \box_use_drop:N \l_@@_cell_box } { } { }
8379 }

```

End of the case when `\l_@@_vpos_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```

8380 {
8381     \pgfextracty \l_tmpa_dim
8382     {
8383         \@@_qpoint:n
8384         {
8385             row - \str_if_eq:eeTF { \l_@@_vpos_block_str } { b } { #3 } { #1 }
8386             - base
8387         }
8388     }
8389     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

8390 \pgfpointanchor
8391 {
8392     \@@_env: - #1 - #2 - block
8393     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8394 }
8395 {
8396     \str_case:on \l_@@_hpos_block_str
8397     {
8398         c { center }

```

```

8399         l { west }
8400         r { east }
8401         j { center }
8402     }
8403 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8404     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8405     \pgfset { inner~sep = \c_zero_dim }
8406     \pgfnode
8407     { rectangle }
8408     {
8409         \str_case:on \l_@@_hpos_block_str
8410         {
8411             c { base }
8412             l { base~west }
8413             r { base~east }
8414             j { base }
8415         }
8416     }
8417     { \box_use_drop:N \l_@@_cell_box } { } { }
8418 }
8419 \endpgfpicture
8420 }
8421 \group_end:
8422 }
8423 \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).

```

8424 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8425 {
8426     \tl_if_empty:NF \l_@@_opacity_tl
8427     {
8428         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8429             {
8430                 \tl_set:Nc \l_@@_fill_tl
8431                 {
8432                     [ opacity = \l_@@_opacity_tl ,
8433                     \tl_tail:o \l_@@_fill_tl
8434                 }
8435             }
8436             {
8437                 \tl_set:Nc \l_@@_fill_tl
8438                 { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8439             }
8440         ]
8441     }

```

The first argument of `\@@_stroke_block:nnn` is a list of options for the rectangle that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

8442 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8443 {
8444     \group_begin:
8445     \tl_clear:N \l_@@_draw_tl
8446     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8447     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8448     \pgfpicture
8449     \pgfrememberpicturepositiononpagetrue
8450     \pgf@relevantforpicturesizefalse

```

```

8451 \tl_if_empty:NF \l_@@_draw_tl
8452 {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8453 \tl_if_eq:NnTF \l_@@_draw_tl { default }
8454 { \CT@arc@ }
8455 { \@@_color:o \l_@@_draw_tl }
8456 }
8457 \pgfsetcornersarced
8458 {
8459 \pgfpoint
8460 { \l_@@_rounded_corners_dim }
8461 { \l_@@_rounded_corners_dim }
8462 }
8463 \@@_cut_on_hyphen:w #2 \q_stop
8464 \int_compare:nNnF { \l_tmpa_tl } > { \c@iRow }
8465 {
8466 \int_compare:nNnF { \l_tmpb_tl } > { \c@jCol }
8467 {
8468 \@@_qpoint:n { row - \l_tmpa_tl }
8469 \dim_set_eq:NN \l_tmpb_dim \pgf@y
8470 \@@_qpoint:n { col - \l_tmpb_tl }
8471 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8472 \@@_cut_on_hyphen:w #3 \q_stop
8473 \int_compare:nNnT { \l_tmpa_tl } > { \c@iRow }
8474 { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8475 \int_compare:nNnT { \l_tmpb_tl } > { \c@jCol }
8476 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8477 \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8478 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8479 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8480 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8481 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8482 \pgfpathrectanglecorners
8483 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8484 { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8485 \dim_compare:nNnTF { \l_@@_rounded_corners_dim } = { \c_zero_dim }
8486 { \pgfusepathqstroke }
8487 { \pgfusepath { stroke } }
8488 }
8489 }
8490 \endpgfpicture
8491 \group_end:
8492 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8493 \keys_define:nn { nicematrix / BlockStroke }
8494 {
8495 color .tl_set:N = \l_@@_draw_tl ,
8496 draw .code:n =
8497 \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8498 draw .default:n = default ,
8499 line-width .dim_set:N = \l_@@_line_width_dim ,
8500 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8501 rounded-corners .default:n = 4 pt
8502 }

```

The first argument of `\@@_vlines_block:nnn` is a list of options for the rules that we will draw. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

8503 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8504 {

```



```

8505 \group_begin:
8506 \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8507 \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8508 \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8509 \@@_cut_on_hyphen:w #2 \q_stop
8510 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8511 \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8512 \@@_cut_on_hyphen:w #3 \q_stop
8513 \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8514 \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8515 \int_step_inline:nnn { \l_@@_tmpd_tl } { \l_tmpb_tl }
8516 {
8517   \use:e
8518   {
8519     \@@_vline:n
8520     {
8521       position = ##1 ,
8522       start = \l_@@_tmpc_tl ,
8523       end = \int_eval:n { \l_tmpa_tl - 1 } ,
8524       total-width = \dim_use:N \l_@@_line_width_dim
8525     }
8526   }
8527 }
8528 \group_end:
8529 }

8530 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8531 {
8532   \group_begin:
8533   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8534   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8535   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8536   \@@_cut_on_hyphen:w #2 \q_stop
8537   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8538   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8539   \@@_cut_on_hyphen:w #3 \q_stop
8540   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8541   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8542   \int_step_inline:nnn { \l_@@_tmpc_tl } { \l_tmpa_tl }
8543   {
8544     \use:e
8545     {
8546       \@@_hline:n
8547       {
8548         position = ##1 ,
8549         start = \l_@@_tmpd_tl ,
8550         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8551         total-width = \dim_use:N \l_@@_line_width_dim
8552       }
8553     }
8554   }
8555   \group_end:
8556 }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax $i-j$) and the third is the last cell of the block (with the same syntax).

```

8557 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8558 {
8559   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8560   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8561   \dim_compare:nNnTF { \l_@@_rounded_corners_dim } > { \c_zero_dim }
8562   { \@@_error:n { borders~forbidden } }

```

```

8563     {
8564         \tl_clear_new:N \l_@@_borders_tikz_tl
8565         \keys_set:no
8566         { nicematrix / OnlyForTikzInBorders }
8567         \l_@@_borders_clist
8568         \@@_cut_on_hyphen:w #2 \q_stop
8569         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8570         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8571         \@@_cut_on_hyphen:w #3 \q_stop
8572         \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8573         \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8574         \@@_stroke_borders_block_i:
8575     }
8576 }

8577 \hook_gput_code:nnn { begindocument } { . }
8578 {
8579     \cs_new_protected:Npe \@@_stroke_borders_block_i:
8580     {
8581         \c_@@_pgfortikzpicture_tl
8582         \@@_stroke_borders_block_ii:
8583         \c_@@_endpgfortikzpicture_tl
8584     }
8585 }

8586 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8587 {
8588     \pgfrememberpicturepositiononpagetrue
8589     \pgf@relevantforpicturesizefalse
8590     \CT@arc@
8591     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8592     \clist_if_in:NnT \l_@@_borders_clist { right }
8593     { \@@_stroke_vertical:n \l_tmpb_tl }
8594     \clist_if_in:NnT \l_@@_borders_clist { left }
8595     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8596     \clist_if_in:NnT \l_@@_borders_clist { bottom }
8597     { \@@_stroke_horizontal:n \l_tmpa_tl }
8598     \clist_if_in:NnT \l_@@_borders_clist { top }
8599     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8600 }

8601 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8602 {
8603     tikz .code:n =
8604         \cs_if_exist:NTF \tikzpicture
8605         { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8606         { \@@_error:n { tikz~in~borders~without~tikz } } ,
8607     tikz .value_required:n = true ,
8608     top .code:n = ,
8609     bottom .code:n = ,
8610     left .code:n = ,
8611     right .code:n = ,
8612     unknown .code:n = \@@_error:n { bad~border }
8613 }

```

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).

```

8614 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8615 {
8616     \@@_qpoint:n \l_@@_tmpc_tl
8617     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8618     \@@_qpoint:n \l_tmpa_tl
8619     \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8620     \@@_qpoint:n { #1 }
8621     \tl_if_empty:NTF \l_@@_borders_tikz_tl

```

```

8622 {
8623   \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8624   \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8625   \pgfusepathqstroke
8626 }
8627 {
8628   \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8629   ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8630 }
8631 }

```

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).

```

8632 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8633 {
8634   \@@_qpoint:n \l_@@_tmpd_tl
8635   \clist_if_in:NnTF \l_@@_borders_clist { left }
8636   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8637   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8638   \@@_qpoint:n \l_tmpb_tl
8639   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8640   \@@_qpoint:n { #1 }
8641   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8642   {
8643     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8644     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8645     \pgfusepathqstroke
8646   }
8647   {
8648     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8649     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8650   }
8651 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

8652 \keys_define:nn { nicematrix / BlockBorders }
8653 {
8654   borders .clist_set:N = \l_@@_borders_clist ,
8655   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8656   rounded-corners .default:n = 4 pt ,
8657   line-width .dim_set:N = \l_@@_line_width_dim
8658 }

```

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.

```

8659 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8660 {
8661   \begin { tikzpicture }
8662   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because #5 is a list of lists.

```

8663   \clist_map_inline:nn { #1 }
8664   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8665 \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8666 \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8667 (
8668 [
8669 xshift = \dim_use:N \l_@@_offset_dim ,
8670 yshift = - \dim_use:N \l_@@_offset_dim
8671 ]
8672 #2 -| #3
8673 )
8674 rectangle
8675 (
8676 [
8677 xshift = - \dim_use:N \l_@@_offset_dim ,
8678 yshift = \dim_use:N \l_@@_offset_dim
8679 ]
8680 \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8681 ) ;
8682 }
8683 \end { tikzpicture }
8684 }
8685 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

8686 \keys_define:nn { nicematrix / SpecialOffset }
8687 { 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.

```

8688 \cs_new_protected:Npn \@@_NullBlock:
8689 { \@@_collect_options:n { \@@_NullBlock_i: } }
8690 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8691 { }

```

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.

```

8692 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
8693 {
8694 \tl_gput_right:Ne \g_@@_pre_code_before_tl
8695 {

```

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`).

```

8696 \@@_subcellcolor:nnnnnn
8697 {
8698 \tl_if_blank:nTF { #1 }
8699 { { \exp_not:n { #2 } } }
8700 { [ #1 ] { \exp_not:n { #2 } } }
8701 }
8702 { \int_use:N \l_@@_first_row_int } % first row of the block
8703 { \int_use:N \l_@@_first_col_int } % first column of the block
8704 { \int_use:N \l_@@_last_row_int } % last row of the block
8705 { \int_use:N \l_@@_last_col_int } % last column of the block
8706 { \int_use:N \l_@@_split_int }
8707 { \int_use:N \l_@@_split_i_int }
8708 }
8709 \ignorespaces
8710 }

8711 \cs_new_protected:Npn \@@_subcellcolor:nnnnnn #1 #2 #3 #4 #5 #6 #7
8712 {
8713 \@@_color_opacity: #1

```

```

8714 \pgfpicture
8715 \pgf@relevantforpicturesizefalse
8716 \@@_qpoint:n { col - #3 }
8717 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8718 \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
8719 \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
8720 \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
8721 \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
8722 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8723 \@@_qpoint:n { row - #2 }
8724 \pgfpathrectanglecorners
8725   { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } { \l_@@_tmpc_dim } }
8726   { \pgfpoint { \l_tmpb_dim } { \pgf@y } }
8727 \pgfusepathqfill
8728 \endpgfpicture
8729 }

```

27 How to draw the dotted lines transparently

```

8730 \cs_set_protected:Npn \@@_renew_matrix:
8731 {
8732   \RenewDocumentEnvironment { pmatrix } { } {
8733     { \pNiceMatrix }
8734     { \endpNiceMatrix }
8735   \RenewDocumentEnvironment { vmatrix } { } {
8736     { \vNiceMatrix }
8737     { \endvNiceMatrix }
8738   \RenewDocumentEnvironment { Vmatrix } { } {
8739     { \VNiceMatrix }
8740     { \endVNiceMatrix }
8741   \RenewDocumentEnvironment { bmatrix } { } {
8742     { \bNiceMatrix }
8743     { \endbNiceMatrix }
8744   \RenewDocumentEnvironment { Bmatrix } { } {
8745     { \BNiceMatrix }
8746     { \endBNiceMatrix }
8747 }

```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8748 \keys_define:nn { nicematrix / Auto }
8749 {
8750   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8751   columns-type .value_required:n = true ,
8752   l .meta:n = { columns-type = l } ,
8753   r .meta:n = { columns-type = r } ,
8754   c .meta:n = { columns-type = c } ,
8755   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8756   delimiters / color .value_required:n = true ,
8757   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8758   delimiters / max-width .default:n = true ,
8759   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8760   delimiters .value_required:n = true ,
8761   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8762   rounded-corners .default:n = 4 pt
8763 }

```

```

8764 \NewDocumentCommand \AutoNiceMatrixWithDelims
8765 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8766 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8767 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8768 {

```

The group is for the protection of the keys.

```

8769 \group_begin:
8770 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8771 \use:e
8772 {
8773 \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8774 { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8775 [ \exp_not:o \l_tmpa_tl ]
8776 }
8777 \int_if_zero:nT { \l_@@_first_row_int }
8778 {
8779 \int_if_zero:nT { \l_@@_first_col_int } { & }
8780 \prg_replicate:nn { #4 - 1 } { & }
8781 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8782 }
8783 \prg_replicate:nn { #3 }
8784 {
8785 \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).

```

8786 \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8787 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8788 }
8789 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
8790 {
8791 \int_if_zero:nT { \l_@@_first_col_int } { & }
8792 \prg_replicate:nn { #4 - 1 } { & }
8793 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8794 }
8795 \end { NiceArrayWithDelims }
8796 \group_end:
8797 }

8798 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
8799 {
8800 \cs_set_protected:cpn { #1 AutoNiceMatrix }
8801 {
8802 \bool_gset_true:N \g_@@_delims_bool
8803 \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8804 \AutoNiceMatrixWithDelims { #2 } { #3 }
8805 }
8806 }

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8807 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8808 {
8809 \group_begin:
8810 \bool_gset_false:N \g_@@_delims_bool
8811 \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8812 \group_end:
8813 }

```

29 The redefinition of the command `\dotfill`

```

8814 \cs_set_eq:NN \@@_old_dotfill: \dotfill
8815 \cs_new_protected:Npn \@@_dotfill:
8816 {

```

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}`).

```

8817 \@@_old_dotfill:
8818 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8819 }

```

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`.

```

8820 \cs_new_protected:Npn \@@_dotfill_i:
8821 {
8822   \dim_compare:nNtT { \box_wd:N \l_@@_cell_box } = { \c_zero_dim }
8823   { \@@_old_dotfill: }
8824 }

```

30 The command `\diagbox`

The command `\diagbox` will be linked to `\diagbox:nn` in the environments of `nicematrix`. However, there are also redefinitions of `\diagbox` in other circumstances.

```

8825 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8826 {
8827   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8828   {
8829     \@@_actually_diagbox:nnnnnn
8830     { \int_use:N \c@iRow }
8831     { \int_use:N \c@jCol }
8832     { \int_use:N \c@iRow }
8833     { \int_use:N \c@jCol }

```

`\g_@@_row_style_tl` contains several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

The command `\@@_if_row_less:nn` is fully expandable and, thus, the instructions will be inserted in the `\g_@@_pre_code_after_tl` only if `\diagbox` is used in a row which is the scope of that chunk of instructions.

```

8834     { \g_@@_row_style_tl \exp_not:n { #1 } }
8835     { \g_@@_row_style_tl \exp_not:n { #2 } }
8836   }

```

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`.

```

8837   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8838   {
8839     { \int_use:N \c@iRow }
8840     { \int_use:N \c@jCol }
8841     { \int_use:N \c@iRow }
8842     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8843     { }
8844   }
8845 }

```

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.

```

8846 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8847 {
8848   \pgfpicture
8849   \pgf@relevantforpicturesizefalse
8850   \pgfrememberpicturepositiononpagetrue
8851   \@@_qpoint:n { row - #1 }
8852   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8853   \@@_qpoint:n { col - #2 }
8854   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8855   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8856   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8857   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8858   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8859   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8860   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8861   {

```

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.

```

8862   \CT@arc@
8863   \pgfsetroundcap
8864   \pgfusepathqstroke
8865 }
8866 \pgfset { inner~sep = 1 pt }
8867 \pgfscope
8868 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8869 \pgfnode { rectangle } { south~west }
8870 {
8871   \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8872   \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8873   \end { minipage }
8874 }
8875 { }
8876 { }
8877 \endpgfscope
8878 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8879 \pgfnode { rectangle } { north~east }
8880 {
8881   \begin { minipage } { 20 cm }
8882   \raggedleft
8883   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8884   \end { minipage }
8885 }
8886 { }
8887 { }
8888 \endpgfpicture
8889 }

```

31 The keyword `\CodeAfter`

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment `{@@-light-syntax}` on p. 90.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
8890 \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 `\`.

```
8891 \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`.

```
8892 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8893 {
8894   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8895   \@@_CodeAfter_iv:n
8896 }
```

We catch the argument of the command `\end` (in `#1`).

```
8897 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8898 {
```

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.

```
8899   \str_if_eq:eeTF { \@currenvir } { #1 }
8900   { \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`.

```
8901   {
8902     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8903     \@@_CodeAfter_ii:n
8904   }
8905 }
```

32 The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_pre_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of column. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```
8906 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8907 {
8908   \pgfpicture
8909   \pgfrememberpicturepositiononpagetrue
8910   \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.

```
8911   \@@_qpoint:n { row - 1 }
8912   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8913   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8914   \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).

```

8915 \bool_if:nTF { #3 }
8916 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8917 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8918 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8919 {
8920   \cs_if_exist:cT
8921   { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8922   {
8923     \pgfpointanchor
8924     { \@@_env: - ##1 - #2 }
8925     { \bool_if:nTF { #3 } { west } { east } }
8926     \dim_set:Nn \l_tmpa_dim
8927     {
8928       \bool_if:nTF { #3 }
8929       { \dim_min:nn }
8930       { \dim_max:nn }
8931       \l_tmpa_dim
8932       { \pgf@x }
8933     }
8934   }
8935 }

```

Now we can put the delimiter with a node of PGF.

```

8936 \pgfset { inner~sep = \c_zero_dim }
8937 \dim_zero:N \nulldelimiterspace
8938 \pgftransformshift
8939 {
8940   \pgfpoint
8941   { \l_tmpa_dim }
8942   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8943 }
8944 \pgfnode
8945 { rectangle }
8946 { \bool_if:nTF { #3 } { east } { west } }
8947 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8948 \nullfont
8949 $ % $
8950 \@@_color:o \l_@@_delimiters_color_tl
8951 \bool_if:nTF { #3 } { \left #1 } { \left . }
8952 \vcenter
8953 {
8954   \nullfont
8955   \hrule \@@height
8956   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8957   \@depth \c_zero_dim
8958   \@width \c_zero_dim
8959 }
8960 \bool_if:nTF { #3 } { \right . } { \right #1 }
8961 $ % $
8962 }
8963 { }
8964 { }
8965 \endpgfpicture
8966 }

```

33 The command `\SubMatrix`

```

8967 \keys_define:nn { nicematrix / sub-matrix }
8968 {
8969   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8970   extra-height .value_required:n = true ,
8971   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8972   left-xshift .value_required:n = true ,
8973   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8974   right-xshift .value_required:n = true ,
8975   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8976   xshift .value_required:n = true ,
8977   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8978   delimiters / color .value_required:n = true ,
8979   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8980   slim .default:n = true ,
8981   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8982   hlines .default:n = all ,
8983   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8984   vlines .default:n = all ,
8985   hvlines .meta:n = { hlines, vlines } ,
8986   hvlines .value_forbidden:n = true
8987 }
8988 \keys_define:nn { nicematrix }
8989 {
8990   SubMatrix .inherit:n = nicematrix / sub-matrix ,
8991   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8992   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8993   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8994 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8995 \keys_define:nn { nicematrix / SubMatrix }
8996 {
8997   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8998   delimiters / color .value_required: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   name .code:n =
9006     \tl_if_empty:nTF { #1 }
9007     { \@@_error:n { Invalid-name } }
9008     {
9009       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
9010       {
9011         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
9012         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
9013         {
9014           \str_set:Nn \l_@@_submatrix_name_str { #1 }
9015           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
9016         }
9017       }
9018       { \@@_error:n { Invalid-name } }
9019     } ,
9020   name .value_required:n = true ,
9021   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
9022   rules .value_required:n = true ,
9023   code .tl_set:N = \l_@@_code_tl ,
9024   code .value_required:n = true ,
9025   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
9026 }

```

```

9027 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
9028 {
9029   \tl_gput_right:Ne \g_@@_pre_code_after_tl
9030   {
9031     \SubMatrix { #1 } { #2 } { #3 } { #4 }
9032     [
9033       delimiters / color = \l_@@_delimiters_color_tl ,
9034       hlines = \l_@@_submatrix_hlines_clist ,
9035       vlines = \l_@@_submatrix_vlines_clist ,
9036       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
9037       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
9038       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
9039       slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
9040       #5
9041     ]
9042   }
9043   \@@_SubMatrix_in_code_before_i { #2 } { #3 }
9044   \ignorespaces
9045 }
9046 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
9047 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9048 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
9049 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
9050 {
9051   \seq_gput_right:Ne \g_@@_submatrix_seq
9052   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

9053   { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
9054   { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
9055   { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
9056   { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
9057 }
9058 }

```

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).

```

9059 \NewDocumentCommand \@@_compute_i_j:nn
9060 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9061 { \@@_compute_i_j:nnnn #1 #2 }
9062 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
9063 {
9064   \def \l_@@_first_i_tl { #1 }
9065   \def \l_@@_first_j_tl { #2 }
9066   \def \l_@@_last_i_tl { #3 }
9067   \def \l_@@_last_j_tl { #4 }
9068   \tl_if_eq:NnT \l_@@_first_i_tl { last }
9069     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
9070   \tl_if_eq:NnT \l_@@_first_j_tl { last }
9071     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
9072   \tl_if_eq:NnT \l_@@_last_i_tl { last }
9073     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
9074   \tl_if_eq:NnT \l_@@_last_j_tl { last }
9075     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
9076 }

```

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`.

```

9077 \hook_gput_code:nnn { begindocument } { . }
9078 {
9079   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
9080   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
9081     { \@@_sub_matrix:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
9082 }
9083 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
9084 {
9085   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

9086   \@@_compute_i_j:nn { #2 } { #3 }
9087   \int_compare:nNt { \l_@@_first_i_tl } = { \l_@@_last_i_tl }
9088   { \def \arraystretch { 1 } }
9089   \bool_lazy_or:nnTF
9090     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9091     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9092     { \@@_error:nn { Construct~too-large } { \SubMatrix } }
9093   {
9094     \str_clear_new:N \l_@@_submatrix_name_str
9095     \keys_set:nn { nicematrix / SubMatrix } { #5 }
9096     \pgfpicture
9097     \pgfrememberpicturepositiononpagetrue
9098     \pgf@relevantforpicturesizefalse
9099     \pgfset { inner~sep = \c_zero_dim }
9100     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9101     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curryfication.

```

9102   \bool_if:NTF \l_@@_submatrix_slim_bool
9103   { \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl } }
9104   { \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int } }
9105   {
9106     \cs_if_exist:cT
9107     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9108     {
9109       \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9110       \dim_compare:nNt { \pgf@x } < { \l_@@_x_initial_dim }
9111       { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9112     }
9113     \cs_if_exist:cT
9114     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9115     {
9116       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9117       \dim_compare:nNt { \pgf@x } > { \l_@@_x_final_dim }
9118       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9119     }
9120   }
9121   \dim_compare:nNtTF { \l_@@_x_initial_dim } = { \c_max_dim }
9122   { \@@_error:nn { Impossible-delimiter } { left } }
9123   {
9124     \dim_compare:nNtTF { \l_@@_x_final_dim } = { - \c_max_dim }

```

```

9125         { \@@_error:nn { Impossible-delimiter } { right } }
9126         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
9127     }
9128     \endpgfpicture
9129 }
9130 \group_end:
9131 \ignorespaces
9132 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

9133 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
9134 {
9135     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
9136     \dim_set:Nn \l_@@_y_initial_dim
9137     {
9138         \fp_to_dim:n
9139         {
9140             \pgf@y
9141             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
9142         }
9143     }
9144     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
9145     \dim_set:Nn \l_@@_y_final_dim
9146     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
9147     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
9148     {
9149         \cs_if_exist:cT
9150         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
9151         {
9152             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
9153             \dim_set:Nn \l_@@_y_initial_dim
9154             { \dim_max:nn { \l_@@_y_initial_dim } { \pgf@y } }
9155         }
9156         \cs_if_exist:cT
9157         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
9158         {
9159             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
9160             \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
9161             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
9162         }
9163     }
9164     \dim_set:Nn \l_tmpa_dim
9165     {
9166         \l_@@_y_initial_dim - \l_@@_y_final_dim +
9167         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
9168     }
9169     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

9170 \group_begin:
9171 \pgfsetlinewidth { 1.1 \arrayrulewidth }
9172 \@@_set_CTarc:o \l_@@_rules_color_tl
9173 \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`.

```

9174 \seq_map_inline:Nn \g_@@_cols_vlism_seq
9175 {
9176     \int_compare:nNnT { \l_@@_first_j_tl } < { ##1 }
9177     {
9178         \int_compare:nNnT
9179         { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }

```

```

9180         {
First, we extract the value of the abscissa of the rule we have to draw.
9181         \@_qpoint:n { col - ##1 }
9182         \pgfpathmoveto { \pgfpoint \pgf@x \l_@_y_initial_dim }
9183         \pgfpathlineto { \pgfpoint \pgf@x \l_@_y_final_dim }
9184         \pgfusepathqstroke
9185     }
9186 }
9187 }

```

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.

```

9188 \str_if_eq:eeTF { \l_@_submatrix_vlines_clist } { all }
9189 { \int_step_inline:nn { \l_@_last_j_tl - \l_@_first_j_tl } }
9190 { \clist_map_inline:Nn \l_@_submatrix_vlines_clist }
9191 {
9192     \bool_lazy_and:nnTF
9193     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
9194     {
9195         \int_compare_p:nNn
9196         { ##1 } < { \l_@_last_j_tl - \l_@_first_j_tl + 1 } }
9197     {
9198         \@_qpoint:n { col - \int_eval:n { ##1 + \l_@_first_j_tl } }
9199         \pgfpathmoveto { \pgfpoint \pgf@x \l_@_y_initial_dim }
9200         \pgfpathlineto { \pgfpoint \pgf@x \l_@_y_final_dim }
9201         \pgfusepathqstroke
9202     }
9203     { \@_error:nmn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
9204 }

```

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.

```

9205 \str_if_eq:eeTF { \l_@_submatrix_hlines_clist } { all }
9206 { \int_step_inline:nn { \l_@_last_i_tl - \l_@_first_i_tl } }
9207 { \clist_map_inline:Nn \l_@_submatrix_hlines_clist }
9208 {
9209     \bool_lazy_and:nnTF
9210     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
9211     {
9212         \int_compare_p:nNn
9213         { ##1 } < { \l_@_last_i_tl - \l_@_first_i_tl + 1 } }
9214     {
9215         \@_qpoint:n { row - \int_eval:n { ##1 + \l_@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

9216 \group_begin:
We compute in \l_tmpa_dim the  $x$ -value of the left end of the rule.
9217 \dim_set:Nn \l_tmpa_dim
9218 { \l_@_x_initial_dim - \l_@_submatrix_left_xshift_dim }
9219 \str_case:nn { #1 }
9220 {
9221     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9222     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
9223     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9224     }
9225     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

9226 \dim_set:Nn \l_tmpb_dim
9227 { \l_@_x_final_dim + \l_@_submatrix_right_xshift_dim }
9228 \str_case:nn { #2 }
9229 {
9230     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }

```

```

9231         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
9232         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9233     }
9234     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9235     \pgfusepathqstroke
9236     \group_end:
9237 }
9238 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
9239 }

```

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).

```

9240 \str_if_empty:NF \l_@@_submatrix_name_str
9241 {
9242     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
9243     \l_@@_x_initial_dim \l_@@_y_initial_dim
9244     \l_@@_x_final_dim \l_@@_y_final_dim
9245 }
9246 \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`.

```

9247 \begin { pgfscope }
9248 \pgftransformshift
9249 {
9250     \pgfpoint
9251     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9252     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9253 }
9254 \str_if_empty:NTF \l_@@_submatrix_name_str
9255 { \@@_node_left:nn #1 { } }
9256 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9257 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9258 \pgftransformshift
9259 {
9260     \pgfpoint
9261     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9262     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9263 }
9264 \str_if_empty:NTF \l_@@_submatrix_name_str
9265 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9266 {
9267     \@@_node_right:nnnn #2
9268     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9269 }

```

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`.

```

9270 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9271 \flag_clear_new:N \l_@@_code_flag
9272 \l_@@_code_tl
9273 }

```

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`.

```

9274 \cs_set_eq:NN \@@_old_pgfpointanchor: \pgfpointanchor

```


The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of TikZ nodes which are computed in an expandable way.

The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by curryfication.

```

9275 \cs_new:Npn \@@_pgfpointanchor:n #1
9276 { \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`).

```

9277 \cs_new:Npn \@@_pgfpointanchor_i:n #1
9278 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }

9279 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9280 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9281 \str_if_empty:nTF { #1 }

```

First, when the name of the name begins with `\tikz@pp@name`.

```

9282 { \@@_pgfpointanchor_iv:w #2 }

```

And now, when there is no `\tikz@pp@name`.

```

9283 { \@@_pgfpointanchor_ii:n { #1 } }
9284 }

```

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`).

```

9285 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
9286 { \@@_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`.

```

9287 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

9288 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
9289 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9290 \str_if_empty:nTF { #2 }

```

First the case where the argument does *not* contain an hyphen.

```

9291 { \@@_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`).

```

9292 { \@@_pgfpointanchor_iii:w { #1 } #2 }
9293 }

```

The following function is for the case when the name contains an hyphen.

```

9294 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9295 {

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9296 \@@_env:
9297 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9298 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9299 }

```

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.

```

9300 \tl_const:Nn \c_@@_integers_alist_tl
9301 {
9302   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9303   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9304   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9305   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9306 }

9307 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9308 {

```

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`.

```

9309   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
9310   {
9311     \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9312   \@@_env: -
9313   \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9314     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9315     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9316   }
9317   {
9318     \str_if_eq:eeTF { #1 } { last }
9319     {
9320       \flag_raise:N \l_@@_code_flag
9321       \@@_env: -
9322       \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9323         { \int_eval:n { \l_@@_last_i_tl + 1 } }
9324         { \int_eval:n { \l_@@_last_j_tl + 1 } }
9325     }
9326     { #1 }
9327   }
9328 }

```

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`).

```

9329 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9330 {
9331   \pgfnode
9332     { rectangle }
9333     { east }
9334     {
9335       \nullfont
9336       $ % $
9337       \@@_color:o \l_@@_delimiters_color_tl
9338       \left #1
9339       \vcenter
9340       {
9341         \nullfont
9342         \hrule \@height \l_tmpa_dim
9343         \@depth \c_zero_dim

```

```

9344         \@width \c_zero_dim
9345     }
9346     \right .
9347     $ % $
9348 }
9349 { #2 }
9350 { }
9351 }

```

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.

```

9352 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9353 {
9354     \pgfnode
9355     { rectangle }
9356     { west }
9357     {
9358         \nullfont
9359         $ % $
9360         \colorlet { current-color } { . }
9361         \@@_color:o \l_@@_delimiters_color_tl
9362         \left .
9363         \vcenter
9364         {
9365             \nullfont
9366             \hrule \@height \l_tmpa_dim
9367                 \@depth \c_zero_dim
9368                 \@width \c_zero_dim
9369         }
9370         \right #1
9371         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9372         ^ { \color { current-color } \smash { #4 } }
9373         $ % $
9374     }
9375     { #2 }
9376     { }
9377 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9378 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9379 {
9380     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9381     \ignorespaces
9382 }
9383 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9384 {
9385     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9386     \ignorespaces
9387 }
9388 \keys_define:nn { nicematrix / Brace }
9389 {
9390     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9391     left-shorten .default:n = true ,
9392     left-shorten .value_forbidden:n = true ,

```

```

9393   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9394   right-shorten .default:n = true ,
9395   right-shorten .value_forbidden:n = true ,
9396   shorten .meta:n = { left-shorten , right-shorten } ,
9397   shorten .value_forbidden:n = true ,
9398   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9399   yshift .value_required:n = true ,
9400   yshift .initial:n = \c_zero_dim ,
9401   color .tl_set:N = \l_tmpa_tl ,
9402   color .value_required:n = true ,
9403   unknown .code:n =
9404     \@@_unknown_key:nn
9405     { nicematrix / Brace }
9406     { Unknown-key-for-Brace }
9407 }

```

#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*.

```

9408 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9409 {
9410   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9411   \@@_compute_i_j:nn { #1 } { #2 }
9412   \bool_lazy_or:nnTF
9413     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9414     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9415     {
9416       \str_if_eq:eeTF { #5 } { under }
9417       { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9418       { \@@_error:nn { Construct-too-large } { \OverBrace } }
9419     }
9420     {
9421       \tl_clear:N \l_tmpa_tl
9422       \keys_set:nn { nicematrix / Brace } { #4 }
9423       \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9424       \pgfpicture
9425       \pgfrememberpicturepositiononpagetrue
9426       \pgf@relevantforpicturesizefalse
9427       \bool_if:NT \l_@@_brace_left_shorten_bool
9428       {
9429         \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9430         \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9431         {
9432           \cs_if_exist:cT
9433             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9434             {
9435               \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9436
9437               \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9438               { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9439             }
9440         }
9441       }
9442       \bool_lazy_or:nnT
9443         { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9444         { \dim_compare_p:nNn { \l_@@_x_initial_dim } = { \c_max_dim } }
9445         {
9446           \@@_qpoint:n { col - \l_@@_first_j_tl }
9447           \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9448         }
9449       \bool_if:NT \l_@@_brace_right_shorten_bool
9450       {

```

```

9451 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9452 \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9453 {
9454   \cs_if_exist:cT
9455   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9456   {
9457     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9458     \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
9459     { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9460   }
9461 }
9462 }
9463 \bool_lazy_or:nnT
9464 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9465 { \dim_compare_p:nNn { \l_@@_x_final_dim } = { - \c_max_dim } }
9466 {
9467   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9468   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9469 }
9470 \pgfset { inner~sep = \c_zero_dim }
9471 \str_if_eq:eeTF { #5 } { under }
9472 { \@@_underbrace_i:n { #3 } }
9473 { \@@_overbrace_i:n { #3 } }
9474 \endpgfpicture
9475 }
9476 \group_end:
9477 }

```

The argument is the text to put above the brace.

```

9478 \cs_new_protected:Npn \@@_overbrace_i:n #1
9479 {
9480   \@@_qpoint:n { row - \l_@@_first_i_tl }
9481   \pgftransformshift
9482   {
9483     \pgfpoint
9484     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9485     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9486   }
9487   \pgfnode
9488   { rectangle }
9489   { south }
9490   {
9491     \vtop
9492     {
9493       \group_begin:
9494       \everycr { }
9495       \halign
9496       {
9497         \hfil ## \hfil \crcr
9498         \bool_if:NTF \l_@@_tabular_bool
9499         { \begin { tabular } { c } #1 \end { tabular } }
9500         { $ \begin { array } { c } #1 \end { array } $ }
9501         \cr
9502         $ % $
9503         \overbrace
9504         {
9505           \hbox_to_wd:nn
9506           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9507           { }
9508         }
9509         $ % $
9510         \cr
9511       }
9512       \group_end:

```

```

9513     }
9514   }
9515   { }
9516   { }
9517 }

```

The argument is the text to put under the brace.

```

9518 \cs_new_protected:Npn \l_@@_underbrace_i:n #1
9519 {
9520   \l_@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9521   \pgftransformshift
9522   {
9523     \pgfpoint
9524     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9525     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9526   }
9527   \pgfnode
9528   { rectangle }
9529   { north }
9530   {
9531     \group_begin:
9532     \everycr { }
9533     \vbox
9534     {
9535       \halign
9536       {
9537         \hfil ## \hfil \crcr
9538         $ % $
9539         \underbrace
9540         {
9541           \hbox_to_wd:nn
9542           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9543           { }
9544         }
9545         $ % $
9546         \cr
9547         \bool_if:NTF \l_@@_tabular_bool
9548         { \begin { tabular } { c } #1 \end { tabular } }
9549         { $ \begin { array } { c } #1 \end { array } $ }
9550         \cr
9551       }
9552     }
9553     \group_end:
9554   }
9555   { }
9556   { }
9557 }

```

35 The commands HBrace et VBrace

The TikZ style `nicematrix/brace` is a TikZ style used to draw the braces created by `\Hbrace` and `\Vbrace`.

We can't load that definition right away because of course, maybe the final user has not yet loaded TikZ (`\Hbrace` and `\Vbrace` are available only when TikZ is loaded and also its library `decorations.pathreplacing`).

```

9558 \AddToHook { package / tikz / after }
9559 {

```

```

9560 \tikzset
9561 {
9562     nicematrix / brace / .style =
9563     {
9564         decoration = { brace , raise = -0.15 em } ,
9565         decorate ,
9566     } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9567     nicematrix / mirrored-brace / .style =
9568     {
9569         nicematrix / brace ,
9570         decoration = mirror ,
9571     }
9572 }
9573 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9574 \keys_define:nn { nicematrix / Hbrace }
9575 {
9576     color .code:n = ,
9577     horizontal-label .code:n = ,
9578     horizontal-labels .code:n = ,
9579     shorten .code:n = ,
9580     shorten-start .code:n = ,
9581     shorten-end .code:n = ,
9582     shorten+ .code:n = ,
9583     shorten-start+ .code:n = ,
9584     shorten-end+ .code:n = ,
9585     shorten~+ .code:n = ,
9586     shorten-start~+ .code:n = ,
9587     shorten-end~+ .code:n = ,
9588     brace-shift .code:n = ,
9589     brace-shift+ .code:n = ,
9590     brace-shift~+ .code:n = ,
9591     unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9592 }

```

Here we need an “fully expandable” command.

```

9593 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9594 {
9595     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9596     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9597     { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9598 }

```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

9599 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9600 {
9601     \int_compare:nNnTF { \c@iRow } < { 2 }
9602     {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9603     \str_if_eq:nnTF { #2 } { * }
9604     {
9605         \bool_set_true:N \l_@@_nullify_dots_bool
9606         \Ldots
9607         [
9608             line-style = nicematrix / brace ,
9609             #1 ,

```

```

9610         up =
9611         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9612     ]
9613 }
9614 {
9615     \Hdotsfor
9616     [
9617         line-style = nicematrix / brace ,
9618         #1 ,
9619         up =
9620         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9621     ]
9622     { #2 }
9623 }
9624 }
9625 {
9626     \str_if_eq:nnTF { #2 } { * }
9627     {
9628         \bool_set_true:N \l_@@_nullify_dots_bool
9629         \Ldots
9630         [
9631             line-style = nicematrix / mirrored-brace ,
9632             #1 ,
9633             down =
9634             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9635         ]
9636     }
9637     {
9638         \Hdotsfor
9639         [
9640             line-style = nicematrix / mirrored-brace ,
9641             #1 ,
9642             down =
9643             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9644         ]
9645         { #2 }
9646     }
9647 }
9648 \keys_set:nn { nicematrix / Hbrace } { #1 }
9649 }

9650 \NewDocumentCommand { \@@_Vbrace } { 0 { } } m m }
9651 {
9652     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9653     { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9654     { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9655 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

9656 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
9657 {
9658     \int_compare:nNnTF { \c@jCol } < { 2 }
9659     {
9660         \str_if_eq:nnTF { #2 } { * }
9661         {
9662             \bool_set_true:N \l_@@_nullify_dots_bool
9663             \Vdots
9664             [
9665                 Vbrace ,
9666                 line-style = nicematrix / mirrored-brace ,
9667                 #1 ,
9668                 down =

```



```

9669         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9670     ]
9671 }
9672 {
9673     \Vdotsfor
9674     [
9675         Vbrace ,
9676         line-style = nicematrix / mirrored-brace ,
9677         #1 ,
9678         down =
9679         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9680     ]
9681     { #2 }
9682 }
9683 }
9684 {
9685     \str_if_eq:nnTF { #2 } { * }
9686     {
9687         \bool_set_true:N \l_@@_nullify_dots_bool
9688         \Vdots
9689         [
9690             Vbrace ,
9691             line-style = nicematrix / brace ,
9692             #1 ,
9693             up =
9694             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9695         ]
9696     }
9697     {
9698         \Vdotsfor
9699         [
9700             Vbrace ,
9701             line-style = nicematrix / brace ,
9702             #1 ,
9703             up =
9704             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9705         ]
9706         { #2 }
9707     }
9708 }
9709 \keys_set:nn { nicematrix / Hbrace } { #1 }
9710 }

```

36 The command TikzEveryCell

```

9711 \bool_new:N \l_@@_not_empty_bool
9712 \bool_new:N \l_@@_empty_bool
9713
9714 \keys_define:nn { nicematrix / TikzEveryCell }
9715 {
9716     not-empty .code:n =
9717     \bool_lazy_or:nnTF
9718     { \l_@@_in_code_after_bool }
9719     { \g_@@_create_cell_nodes_bool }
9720     { \bool_set_true:N \l_@@_not_empty_bool }
9721     { \@@_error:n { detection-of-empty-cells } } ,
9722     not-empty .value_forbidden:n = true ,
9723     empty .code:n =
9724     \bool_lazy_or:nnTF

```

```

9725     { \l_@@_in_code_after_bool }
9726     { \g_@@_create_cell_nodes_bool }
9727     { \bool_set_true:N \l_@@_empty_bool }
9728     { \@@_error:n { detection~of~empty~cells } } } ,
9729     empty .value_forbidden:n = true ,
9730     unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
9731 }
9732
9733
9734 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9735 {
9736   \IfPackageLoadedTF { tikz }
9737   {
9738     \group_begin:
9739     \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.

```

9740     \tl_set:Nn \l_tmpa_tl { { #2 } }
9741     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9742     { \@@_for_a_block:nnnnn ##1 }
9743     \@@_all_the_cells:
9744     \group_end:
9745   }
9746   { \@@_error:n { TikzEveryCell~without~tikz } }
9747 }
9748
9749
9750 \cs_new_protected:Nn \@@_all_the_cells:
9751 {
9752   \int_step_inline:nn \c@iRow
9753   {
9754     \int_step_inline:nn \c@jCol
9755     {
9756       \cs_if_exist:cF { cell - ##1 - ####1 }
9757       {
9758         \clist_if_in:NcF \l_@@_corners_cells_clist
9759         { ##1 - ####1 }
9760         {
9761           \bool_set_false:N \l_tmpa_bool
9762           \cs_if_exist:cTF
9763           { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 }
9764           {
9765             \bool_if:NF \l_@@_empty_bool
9766             { \bool_set_true:N \l_tmpa_bool }
9767           }
9768           {
9769             \bool_if:NF \l_@@_not_empty_bool
9770             { \bool_set_true:N \l_tmpa_bool }
9771           }
9772           \bool_if:NT \l_tmpa_bool
9773           {
9774             \@@_block_tikz:nnnnn
9775             \l_tmpa_tl { ##1 } { ####1 } { ##1 } { ####1 }
9776           }
9777         }
9778       }
9779     }
9780   }
9781 }
9782
9783 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9784 {
9785   \bool_if:NF \l_@@_empty_bool

```

```

9786     {
9787         \@@_block_tikz:nnnn
9788         \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9789     }
9790     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9791 }
9792
9793 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9794 {
9795     \int_step_inline:nnn { #1 } { #3 }
9796     {
9797         \int_step_inline:nnn { #2 } { #4 }
9798         { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9799     }
9800 }

```

37 The command \CreateBlocksInColumn

```

9801 \cs_new_protected:Npn \@@_create_blocks_in_col:n #1
9802 { \clist_gput_right:Nn \g_@@_cbic_clist { #1 } }
9803
9804 \cs_new_protected:Npn \@@_cbic:
9805 {
9806     \clist_map_inline:Nn \g_@@_cbic_clist
9807     {
9808         \cs_set:cpn
9809         {
9810             pgf @ sh @ ns @ \@@_env:
9811             - \int_eval:n { \c@iRow + 1 } - ##1 }
9812         { rien }
9813     }
9814
9815     \l_tmpa_int will be the first row of the block being created.
9816     \int_set:Nn \l_tmpa_int { 1 }
9817     \int_step_inline:nn { \c@iRow + 1 }
9818     {
9819         \cs_if_exist:cT
9820         { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 }
9821         {
9822             \int_compare:nNnT { #####1 } > { \l_tmpa_int + 1 }
9823             {
9824                 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
9825                 {
9826                     { \int_use:N \l_tmpa_int }
9827                     { ##1 }
9828                     { \int_eval:n { #####1 - 1 } }
9829                     { ##1 }
9830                 }
9831             }
9832             \int_set:Nn \l_tmpa_int { #####1 }
9833         }
9834     }
9835     \clist_gclear:N \g_@@_cbic_clist
9836 }

```

38 The command \ShowCellNames

```

9835 \NewDocumentCommand \@@_ShowCellNames { }
9836 {
9837     \bool_if:NT \l_@@_in_code_after_bool

```

```

9838 {
9839   \pgfpicture
9840   \pgfrememberpicturepositiononpagetrue
9841   \pgf@relevantforpicturesizefalse
9842   \pgfpathrectanglecorners
9843     { \@@_qpoint:n { 1 } }
9844     {
9845       \@@_qpoint:n
9846       { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
9847     }
9848   \pgfsetfillopacity { 0.75 }
9849   \pgfsetfillcolor { white }
9850   \pgfusepathqfill
9851   \endpgfpicture
9852 }
9853 \dim_gzero_new:N \g_@@_tmpc_dim
9854 \dim_gzero_new:N \g_@@_tmpd_dim
9855 \dim_gzero_new:N \g_@@_tmpe_dim
9856 \int_step_inline:nn { \c@iRow }
9857 {
9858   \bool_if:NTF \l_@@_in_code_after_bool
9859   {
9860     \pgfpicture
9861     \pgfrememberpicturepositiononpagetrue
9862     \pgf@relevantforpicturesizefalse
9863   }
9864   { \begin { pgfpicture } }
9865   \@@_qpoint:n { row - ##1 }
9866   \dim_set_eq:NN \l_tmpa_dim \pgf@y
9867   \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9868   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9869   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9870   \bool_if:NTF \l_@@_in_code_after_bool
9871   { \endpgfpicture }
9872   { \end { pgfpicture } }
9873   \int_step_inline:nn { \c@jCol }
9874   {
9875     \hbox_set:Nn \l_tmpa_box
9876     {
9877       \normalfont \Large \sffamily \bfseries
9878       \bool_if:NTF \l_@@_in_code_after_bool
9879       { \color { red } }
9880       { \color { red ! 50 } }
9881       ##1 - ####1
9882     }
9883     \bool_if:NTF \l_@@_in_code_after_bool
9884     {
9885       \pgfpicture
9886       \pgfrememberpicturepositiononpagetrue
9887       \pgf@relevantforpicturesizefalse
9888     }
9889     { \begin { pgfpicture } }
9890     \@@_qpoint:n { col - ####1 }
9891     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9892     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9893     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9894     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9895     \bool_if:NTF \l_@@_in_code_after_bool
9896     { \endpgfpicture }
9897     { \end { pgfpicture } }
9898     \fp_set:Nn \l_tmpa_fp
9899     {
9900       \fp_min:nn

```

```

9901         {
9902             \fp_min:nn
9903             { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9904             { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9905         }
9906         { 1.0 }
9907     }
9908     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9909     \pgfpicture
9910     \pgfrememberpicturepositiononpagetrue
9911     \pgf@relevantforpicturesizefalse
9912     \pgftransformshift
9913     {
9914         \pgfpoint
9915         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9916         { \dim_use:N \g_tmpa_dim }
9917     }
9918     \pgfnode
9919     { rectangle }
9920     { center }
9921     { \box_use:N \l_tmpa_box }
9922     { }
9923     { }
9924     \endpgfpicture
9925 }
9926 }
9927 }

```

39 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.

```

9928 \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.

```

9929 \bool_new:N \g_@@_footnote_bool
9930 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9931 {
9932     You-have-used-the-key~' \l_keys_key_str '~when-loading-nicematrix~
9933     but-that-key-is-unknown. \\
9934     It-will-be-ignored. \\
9935     For-a-list-of-the-available-keys,~type-H~<return>.
9936 }
9937 {
9938     The-available-keys-are~(in-alphabetic-order):~
9939     footnote,~
9940     footnotehyper,~
9941     messages-for-Overleaf,~
9942     renew-dots~and~
9943     renew-matrix.
9944 }
9945 \keys_define:nn { nicematrix }
9946 {
9947     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,

```

```

9948   renew-dots .value_forbidden:n = true ,
9949   renew-matrix .code:n = \@_renew_matrix: ,
9950   renew-matrix .value_forbidden:n = true ,
9951   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9952   footnote .bool_set:N = \g_@@_footnote_bool ,
9953   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9954   unknown .code:n = \@_error:n { Unknown-key-for-package }
9955 }
9956 \ProcessKeyOptions

9957 \@_msg_new:n { footnote-with-footnotehyper-package }
9958 {
9959   You-can't-use-the-option-'footnote'~because-the-package~
9960   footnotehyper-has-already-been-loaded.~
9961   If-you-want,-you-can-use-the-option-'footnotehyper'~and-the-footnotes~
9962   within-the-environments-of-nicematrix-will-be-extracted-with-the-tools~
9963   of-the-package-footnotehyper.\\
9964   The-package-footnote-won't-be-loaded.
9965 }

9966 \@_msg_new:n { footnotehyper-with-footnote-package }
9967 {
9968   You-can't-use-the-option-'footnotehyper'~because-the-package~
9969   footnote-has-already-been-loaded.~
9970   If-you-want,-you-can-use-the-option-'footnote'~and-the-footnotes~
9971   within-the-environments-of-nicematrix-will-be-extracted-with-the-tools~
9972   of-the-package-footnote.\\
9973   The-package-footnotehyper-won't-be-loaded.
9974 }

9975 \bool_if:NT \g_@@_footnote_bool
9976 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9977   \IfClassLoadedTF { beamer }
9978   { \bool_set_false:N \g_@@_footnote_bool }
9979   {
9980     \IfPackageLoadedTF { footnotehyper }
9981     { \@_error:n { footnote-with-footnotehyper-package } }
9982     { \usepackage { footnote } }
9983   }
9984 }

9985 \bool_if:NT \g_@@_footnotehyper_bool
9986 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9987   \IfClassLoadedTF { beamer }
9988   { \bool_set_false:N \g_@@_footnote_bool }
9989   {
9990     \IfPackageLoadedTF { footnote }
9991     { \@_error:n { footnotehyper-with-footnote-package } }
9992     { \usepackage { footnotehyper } }
9993   }
9994   \bool_set_true:N \g_@@_footnote_bool
9995 }

```

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}`.

40 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.

```
9996 \bool_new:N \l_@@_underscore_loaded_bool
9997 \IfPackageLoadedT { underscore }
9998 { \bool_set_true:N \l_@@_underscore_loaded_bool }

9999 \hook_gput_code:nnn { begindocument } { . }
10000 {
10001   \bool_if:NF \l_@@_underscore_loaded_bool
10002   {
10003     \IfPackageLoadedT { underscore }
10004     { \@@_error:n { underscore-after-nicematrix } }
10005   }
10006 }
```

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.

```
10007 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
10008 {
10009   \str_if_eq:eeTF
10010   { \str_item:Nn \l_keys_key_str { \str_count:N \l_keys_key_str } }
10011   { + }
10012   {
10013     \str_set:Ne \l_tmpa_str
10014     { \str_range:Nnn \l_keys_key_str { 1 } { \str_count:N \l_keys_key_str - 1 } }
10015     \bool_set_false:N \l_tmpa_bool
10016     \clist_map_inline:nn { #1 }
10017     {
10018       \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10019       {
10020         \@@_error:n { key~without~+~exists }
10021         \bool_set_true:N \l_tmpa_bool
10022         \clist_map_break:
10023       }
10024     }
10025     \bool_if:NF \l_tmpa_bool
10026     {
10027       \str_set:Ne \l_keys_key_str { \tl_trim_right_spaces:V \l_tmpa_str }
10028       \@@_unknown_key_i:nn { #1 } { #2 }
10029     }
10030   }
10031   { \@@_unknown_key_i:nn { #1 } { #2 } }
10032 }
```

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.

```
10033 \cs_new_protected:Npn \@@_unknown_key_i:nn #1 #2
```

```

10034 {
10035   \str_set_eq:NN \l_tmpa_str \l_keys_key_str
10036   \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
10037   \str_set:Nx \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
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~with~normal~form~exists }
10044         \bool_set_true:N \l_tmpa_bool
10045         \clist_map_break:
10046       }
10047     }
10048   \bool_if:NF \l_tmpa_bool { \@@_error:n { #2 } }
10049 }
10050 \@@_msg_new:nn { key~without~+~exists }
10051 {
10052   The~key~'\tl_trim_right_spaces:V \l_tmpa_str'~exists~but~does~not~accept~an~
10053   additive~syntax~(with~+=).\
10054   It~will~be~ignored.\
10055 }
10056 \@@_msg_new:nn { key~with~normal~form~exists }
10057 {
10058   The~key~'\l_keys_key_str'~does~not~exists.\
10059   It~will~be~ignored.\
10060   Maybe~you~want~to~use~the~key~'\l_tmpa_str'.
10061 }
10062 \str_const:Nx \c_@@_available_keys_str
10063 {
10064   \bool_if:nT { ! \g_@@_messages_for_Overleaf_bool }
10065   { For~a~list~of~the~available~keys,~type~H~<return>. }
10066 }
10067 \seq_new:N \g_@@_types_of_matrix_seq
10068 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
10069 {
10070   NiceMatrix ,
10071   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
10072 }
10073 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
10074 { \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`.

```

10075 \cs_new_protected:Npn \@@_err_too_many_cols:
10076 {
10077   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
10078   { \@@_fatal:nn { too-many-cols-for-array } }
10079   \int_compare:nNnT { \l_@@_last_col_int } = { -2 }
10080   { \@@_fatal:n { too-many~cols~for~matrix } }
10081   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
10082   { \@@_fatal:n { too-many~cols~for~matrix } }
10083   \bool_if:NF \l_@@_last_col_without_value_bool
10084   { \@@_fatal:n { too-many~cols~for~matrix~with~last~col } }
10085 }

```

The following command must *not* be protected since it's used in an error message.

```

10086 \cs_new:Npn \@@_message_hdotsfor:
10087 {

```



```

10088 \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
10089 { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
10090 \token_to_str:N \Hbrace \ is~incorrect. }
10091 }

10092 \cs_new_protected:Npn \@@_Hline_in_cell:
10093 { \@@_fatal:n { Misuse~of~Hline } }

10094 \@@_msg_new:nn { Misuse~of~Hline }
10095 {
10096   Misuse~of~Hline. \\
10097   Error~in~your~row~ \int_eval:n { \c@iRow }. \\
10098   \token_to_str:N \Hline\ must~be~used~only~at~the~beginning~of~a~row.\\
10099   That~error~is~fatal.
10100 }

10101 \@@_msg_new:nn { hvlines,~rounded~corners~and~corners }
10102 {
10103   Incompatible~options.\\
10104   You~should~not~use~'hvlines',~'rounded~corners'~and~'corners'~at~the~same~time.\\
10105   The~output~will~not~be~reliable.
10106 }

10107 \@@_msg_new:nn { Body~alone }
10108 {
10109   \token_to_str:N \Body\ alone. \\
10110   You~have~used~\token_to_str:N \Body\ without~\token_to_str:N \CodeBefore.\\
10111   That~error~is~fatal.
10112 }

10113 \@@_msg_new:nn { cellcolor~in~Block }
10114 {
10115   Bad~use~of~\token_to_str:N \cellcolor \\
10116   You~can't~use~\token_to_str:N \cellcolor\ in~\token_to_str:N \Block\
10117   (except~in~a~sub~block).\\
10118   That~command~will~be~ignored.
10119 }

10120 \@@_msg_new:nn { rowcolor~in~Block }
10121 {
10122   Bad~use~of~\token_to_str:N \rowcolor \\
10123   You~can't~use~\token_to_str:N \rowcolor\ in~\token_to_str:N \Block.\\
10124   That~command~will~be~ignored.
10125 }

10126 \@@_msg_new:nn { key~color~inside }
10127 {
10128   Deleted~key.\\
10129   The~key~'color~inside'~(and~its~alias~'colortbl~like')~has~been~deleted~in
10130   ~'nicematrix'~and~must~not~be~used.\\
10131   This~error~is~fatal.
10132 }

10133 \@@_msg_new:nn { invalid~weight }
10134 {
10135   Unknown~key.\\
10136   The~key~'\l_keys_key_str '~of~your~column~X~is~unknown~and~will~be~ignored.
10137 }

10138 \@@_msg_new:nn { last~col~not~used }
10139 {
10140   Column~not~used.\\
10141   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
10142   in~your~\@@_full_name_env: .~
10143   However,~you~can~go~on.
10144 }

10145 \@@_msg_new:nn { too~many~cols~for~matrix~with~last~col }
10146 {

```

```

10147     Too-many-columns.\\
10148     In~the~row~ \int_eval:n { \c@iRow },~
10149     you~try~to~use~more~columns~
10150     than~allowed~by~your~ \@@_full_name_env: .
10151     \@@_message_hdotsfor: \
10152     The~maximal~number~of~columns~is~ \int_eval:n { \l_@@_last_col_int - 1 }~
10153     (plus~the~exterior~columns).~This~error~is~fatal.
10154   }

10155 \@@_msg_new:nn { too-many-cols-for-matrix }
10156 {
10157   Too-many-columns.\\
10158   In~the~row~ \int_eval:n { \c@iRow },~
10159   you~try~to~use~more~columns~than~allowed~by~your~ \@@_full_name_env: .
10160   \@@_message_hdotsfor: \
10161   Recall~that~the~maximal~number~of~columns~for~a~matrix~
10162   (excepted~the~potential~exterior~columns)~is~fixed~by~the~
10163   LaTeX~counter~'MaxMatrixCols'.~
10164   Its~current~value~is~ \int_use:N \c@MaxMatrixCols \
10165   (use~ \token_to_str:N \setcounter \ to~change~that~value).~
10166   This~error~is~fatal.
10167 }

10168 \@@_msg_new:nn { too-many-cols-for-array }
10169 {
10170   Too-many-columns.\\
10171   In~the~row~ \int_eval:n { \c@iRow },~
10172   ~you~try~to~use~more~columns~than~allowed~by~your~
10173   \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal~number~of~columns~is~
10174   \int_use:N \g_@@_static_num_of_col_int \
10175   \bool_if:nT
10176     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
10177     { (plus~the~exterior~ones)~}
10178   since~the~preamble~is~' \g_@@_user_preamble_tl '.\\
10179   This~error~is~fatal.
10180 }

10181 \@@_msg_new:nn { columns-not-used }
10182 {
10183   Columns~not~used.\\
10184   The~preamble~of~your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '.~
10185   It~announces~ \int_use:N \g_@@_static_num_of_col_int \
10186   columns~but~you~only~used~ \int_use:N \c@jCol .\\
10187   The~columns~you~did~not~used~won't~be~created.\\
10188   You~won't~have~similar~warning~till~the~end~of~the~document.
10189 }

10190 \@@_msg_new:nn { empty-preamble }
10191 {
10192   Empty~preamble.\\
10193   The~preamble~of~your~ \@@_full_name_env: \ is~empty.\\
10194   This~error~is~fatal.
10195 }

10196 \@@_msg_new:nn { in~first~col }
10197 {
10198   Erroneous~use.\\
10199   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
10200   That~command~will~be~ignored.
10201 }

10202 \@@_msg_new:nn { in~last~col }
10203 {
10204   Erroneous~use.\\
10205   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
10206   That~command~will~be~ignored.
10207 }

```

```

10208 \@@_msg_new:nn { in~first~row }
10209 {
10210   Erroneous~use.\\
10211   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
10212   That~command~will~be~ignored.
10213 }
10214 \@@_msg_new:nn { in~last~row }
10215 {
10216   Erroneous~use.\\
10217   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
10218   That~command~will~be~ignored.
10219 }
10220 \@@_msg_new:nn { TopRule~without~booktabs }
10221 {
10222   Erroneous~use.\\
10223   You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
10224   That~command~will~be~ignored.
10225 }
10226 \@@_msg_new:nn{ rotate~in~p~col }
10227 {
10228   \token_to_str:N \rotate\ forbidden.\\
10229   You~should~not~use~\token_to_str:N \rotate\ in~a~column~of~type~'p',~
10230   'b',~'m'\IfPackageLoadedTF { varwidth } { ,~'X'~or~'V' } { ~or~'X'}.~
10231   If~you~go~on,~maybe~you~won't~have~the~expected~output.
10232 }
10233 \@@_msg_new:nn { TopRule~without~tikz }
10234 {
10235   Erroneous~use.\\
10236   You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
10237   That~command~will~be~ignored.
10238 }
10239 \@@_msg_new:nn { caption~outside~float }
10240 {
10241   Key~caption~forbidden.\\
10242   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
10243   environment~(such~as~\{table\}).~This~key~will~be~ignored.
10244 }
10245 \@@_msg_new:nn { short~caption~without~caption }
10246 {
10247   You~should~not~use~the~key~'short~caption'~without~'caption'.~
10248   However,~your~'short~caption'~will~be~used~as~'caption'.
10249 }
10250 \@@_msg_new:nn { double~closing~delimiter }
10251 {
10252   Double~delimiter.\\
10253   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
10254   delimiter.~This~delimiter~will~be~ignored.
10255 }
10256 \@@_msg_new:nn { delimiter~after~opening }
10257 {
10258   Double~delimiter.\\
10259   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
10260   delimiter.~That~delimiter~will~be~ignored.
10261 }
10262 \@@_msg_new:nn { bad~option~for~line~style }
10263 {
10264   Bad~line~style.\\
10265   Since~you~haven't~loaded~TikZ,~the~only~value~you~can~give~to~'line~style'~
10266   is~'standard'.~That~key~will~be~ignored.
10267 }

```

```

10268 \@@_msg_new:nn { corners-with-no-cell-nodes }
10269 {
10270   Incompatible-keys.\
10271   You-can't-use-the-key~'corners'~here~because~the~key~'no-cell-nodes'~
10272   is~in~force.\
10273   If~you~go~on,~that~key~will~be~ignored.
10274 }
10275 \@@_msg_new:nn { extra-nodes-with-no-cell-nodes }
10276 {
10277   Incompatible-keys.\
10278   You-can't~create~'extra-nodes'~here~because~the~key~'no-cell-nodes'~
10279   is~in~force.\
10280   If~you~go~on,~those~extra~nodes~won't~be~created.
10281 }
10282 \@@_msg_new:nn { Identical-notes-in-caption }
10283 {
10284   Identical~tabular~notes.\
10285   You-can't~put~several~notes~with~the~same~content~in~
10286   \token_to_str:N \caption \ (but~you~can~in~the~main~tabular).\
10287   If~you~go~on,~the~output~will~probably~be~erroneous.
10288 }
10289 \@@_msg_new:nn { tabularnote~below~the~tabular }
10290 {
10291   \token_to_str:N \tabularnote \ forbidden\
10292   You-can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
10293   of~your~tabular~because~the~caption~will~be~composed~below~
10294   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
10295   key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .\
10296   Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
10297   no~similar~error~will~raised~in~this~document.
10298 }
10299 \@@_msg_new:nn { Unknown~key~for~rules }
10300 {
10301   Unknown~key.\
10302   There~is~only~two~keys~available~here:~width~and~color.\
10303   Your~key~' \l_keys_key_str ' ~will~be~ignored.
10304 }
10305 \@@_msg_new:nn { Unknown~key~for~Hbrace }
10306 {
10307   Unknown~key.\
10308   You~have~used~the~key~' \l_keys_key_str ' ~but~the~only~
10309   keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
10310   and~ \token_to_str:N \Vbrace \ are:~'brace-shift(+)',~'color',~
10311   'horizontal-label(s)',~'shorten'~'shorten-end'~
10312   and~'shorten-start'~.\
10313   That~error~is~fatal.
10314 }
10315 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
10316 {
10317   Unknown~key.\
10318   There~is~only~two~keys~available~here:~
10319   'empty'~and~'not-empty'~.\
10320   Your~key~' \l_keys_key_str ' ~will~be~ignored.
10321 }
10322 \@@_msg_new:nn { Unknown~key~for~rotate }
10323 {
10324   Unknown~key.\
10325   The~only~key~available~here~is~'c'~.\
10326   Your~key~' \l_keys_key_str ' ~will~be~ignored.
10327 }

```

```

10328 \@@_msg_new:nnn { Unknown~key~for~custom~line }
10329 {
10330   Unknown~key.\\
10331   The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
10332   It~you~go~on,~you~will~probably~have~other~errors. \\
10333   \c_@@_available_keys_str
10334 }
10335 {
10336   The~available~keys~are~(in~alphabetic~order):~
10337   ccommand,~
10338   color,~
10339   command,~
10340   dotted,~
10341   letter,~
10342   multiplicity,~
10343   sep-color,~
10344   tikz,~and~total~width.
10345 }
10346 \@@_msg_new:nnn { Unknown~key~for~xdots }
10347 {
10348   Unknown~key.\\
10349   The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
10350   \c_@@_available_keys_str
10351 }
10352 {
10353   The~available~keys~are~(in~alphabetic~order):~
10354   'color',~
10355   'horizontal(s)-labels',~
10356   'inter',~
10357   'line-style',~
10358   'nullify',~
10359   'radius',~
10360   'shorten',~
10361   'shorten-end'~and~'shorten-start'.
10362 }
10363 \@@_msg_new:nn { Unknown~key~for~rowcolors }
10364 {
10365   Unknown~key.\\
10366   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
10367   (and~you~try~to~use~' \l_keys_key_str '~)\\
10368   That~key~will~be~ignored.
10369 }
10370 \@@_msg_new:nn { label~without~caption }
10371 {
10372   You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
10373   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
10374 }
10375 \@@_msg_new:nn { W~warning }
10376 {
10377   Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
10378   (row~ \int_use:N \c@iRow ).
10379 }
10380 \@@_msg_new:nn { Construct~too~large }
10381 {
10382   Construct~too~large.\\
10383   Your~command~ \token_to_str:N #1
10384   can't~be~drawn~because~your~matrix~is~too~small.\\
10385   That~command~will~be~ignored.
10386 }
10387 \@@_msg_new:nn { underscore~after~nicematrix }
10388 {

```

```

10389 Problem-with~'underscore'.\\
10390 The-package~'underscore'~should-be-loaded-before~'nicematrix'.~
10391 You-can-go-on-but-you-won't-be-able-to-write-something-such-as:\\
10392 ' \token_to_str:N \Cdots \token_to_str:N _
10393 \{ n \token_to_str:N \text \{ ~times \} \}' .
10394 }
10395 \@@_msg_new:nn { ampersand-in~light-syntax }
10396 {
10397     Ampersand~forbidden.\\
10398     You-can't-use-an-ampersand~( \token_to_str:N &)~to-separate-columns~because~
10399     ~the-key~'light-syntax'~is-in-force.~This-error-is-fatal.
10400 }
10401 \@@_msg_new:nn { double-backslash-in~light-syntax }
10402 {
10403     Double~backslash~forbidden.\\
10404     You-can't-use~ \token_to_str:N \\
10405     ~to-separate-rows~because~the-key~'light-syntax'~
10406     is-in~force.~You-must-use~the-character~' \l_@@_end_of_row_tl '~
10407     (set~by~the-key~'end-of-row').~This-error-is-fatal.
10408 }
10409 \@@_msg_new:nn { hlines-with~color }
10410 {
10411     Incompatible-keys.\\
10412     You-can't-use~the-keys~'hlines',~'vlines'~or~'hvlines'~for~a~
10413     \token_to_str:N \Block \ when~the-key~'color'~or~'draw'~is-used.\\
10414     However,~you-can-put~several-commands~ \token_to_str:N \Block.\\
10415     Your-key~will-be-discarded.
10416 }
10417 \@@_msg_new:nn { bad-value-for~baseline }
10418 {
10419     Bad-value-for~baseline.\\
10420     The-value-given-to~'baseline'~( \int_use:N \l_tmpa_int )~is-not~
10421     valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int\ and~
10422     \int_use:N \g_@@_row_total_int \ or~equal-to~'t',~'c'~or~'b'~or~of~
10423     the-form~'line-i'.\\
10424     A-value-of~1~will-be-used.
10425 }
10426 \@@_msg_new:nn { bad-value-for~baseline-line }
10427 {
10428     Bad-value-for~baseline-with~line.\\
10429     The-value-given-to~'baseline'~( \int_use:N \l_tmpa_int )~is-not~
10430     valid.~The-number-of~the-line-must-be-between~1~and~
10431     \int_eval:n { \c@iRow + 1 } \\
10432     A-value-of~'line-1'~will-be-used.
10433 }
10434 \@@_msg_new:nn { detection-of~empty~cells }
10435 {
10436     Problem-with~'not-empty'\\
10437     For-technical-reasons,~you-must-activate~
10438     'create-cell-nodes'~in~ \token_to_str:N \CodeBefore \
10439     in~order~to~use~the-key~' \l_keys_key_str '.\\
10440     That-key~will-be-ignored.
10441 }
10442 \@@_msg_new:nn { siunitx-not~loaded }
10443 {
10444     siunitx-not~loaded\\
10445     You-can't-use~the-columns~'S'~because~'siunitx'~is-not~loaded.\\
10446     That-error-is-fatal.
10447 }
10448 \@@_msg_new:nn { Invalid~name }

```

```

10449 {
10450   Invalid-name.\
10451   You-can't-give-the-name~' \l_keys_value_tl '~to-a~ \token_to_str:N
10452   \SubMatrix \ of~your~ \@@_full_name_env: .\
10453   A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\
10454   This-key-will-be-ignored.
10455 }
10456 \@@_msg_new:nn { Hbrace-not-allowed }
10457 {
10458   Command-not-allowed.\
10459   You-can't-use-the-command~ \token_to_str:N #1
10460   because-you-have-not-loaded~
10461   \IfPackageLoadedTF { tikz }
10462     { the-TikZ-library~'decorations.pathreplacing'~Use~ }
10463     { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
10464   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \
10465   That-command-will-be-ignored.
10466 }
10467 \@@_msg_new:nn { Vbrace-not-allowed }
10468 {
10469   Command-not-allowed.\
10470   You-can't-use-the-command~ \token_to_str:N \Vbrace \
10471   because-you-have-not-loaded-TikZ~
10472   and-the-TikZ-library~'decorations.pathreplacing'.\
10473   Use: ~\token_to_str:N \usepackage \{tikz\}~
10474   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \
10475   That-command-will-be-ignored.
10476 }
10477 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
10478 {
10479   Wrong-line.\
10480   You-try-to-draw-a-#1~line-of-number~'#2'~in-a~
10481   \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but-that~
10482   number-is-not-valid.~It-will-be-ignored.
10483 }
10484 \@@_msg_new:nn { Impossible-delimiter }
10485 {
10486   Impossible-delimiter.\
10487   It's-impossible-to-draw-the-#1-delimiter-of~your~
10488   \token_to_str:N \SubMatrix \ because-all-the-cells-are-empty~
10489   in-that-column.
10490   \bool_if:NT \l_@@_submatrix_slim_bool
10491     { ~Maybe-you-should-try-without-the-key~'slim'. } \
10492   This~ \token_to_str:N \SubMatrix \ will-be-ignored.
10493 }
10494 \@@_msg_new:nnn { width-without-X-columns }
10495 {
10496   You-have-used-the-key~'width'~but-you-have-put-no~'X'~column-in~
10497   the-preamble~(' \g_@@_user_preamble_tl '~)~of~your~ \@@_full_name_env: .\
10498   That-key-will-be-ignored.
10499 }
10500 {
10501   This-message-is-the-message~'width-without-X-columns'~
10502   of-the-module~'nicematrix'.~
10503   The-experimented-users-can-disable-that-message-with~
10504   \token_to_str:N \msg_redirect_name:nnn .\
10505 }
10506
10507 \@@_msg_new:nn { key-multiplicity-with-dotted }
10508 {
10509   Incompatible-keys. \

```

```

10510     You-have-used-the-key~'multiplicity'~with~the~key~'dotted'~
10511     in~a~'custom-line'~.~They~are~incompatible.~\\
10512     The-key~'multiplicity'~will~be~discarded.
10513 }

10514 \@@_msg_new:nn { empty~environment }
10515 {
10516     Empty~environment.\\
10517     Your~ \@@_full_name_env: \ is~empty.~This~error~is~fatal.
10518 }

10519 \@@_msg_new:nn { No~letter~and~no~command }
10520 {
10521     Erroneous~use.\\
10522     Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
10523     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10524     '~'ccommand'~(to~draw~horizontal~rules).\\
10525     However,~you~can~go~on.
10526 }

10527 \@@_msg_new:nn { Forbidden~letter }
10528 {
10529     Forbidden~letter.\\
10530     You~can't~use~the~letter~'#1'~for~a~customized~line.~
10531     It~will~be~ignored.\\
10532     The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10533 }

10534 \@@_msg_new:nn { Several~letters }
10535 {
10536     Wrong~name.\\
10537     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10538     have~used~' \l_@@_letter_str ').\\
10539     It~will~be~ignored.
10540 }

10541 \@@_msg_new:nn { Delimiter~with~small }
10542 {
10543     Delimiter~forbidden.\\
10544     You~can't~put~a~delimiter~in~the~preamble~of~your~
10545     \@@_full_name_env: \
10546     because~the~key~'small'~is~in~force.\\
10547     This~error~is~fatal.
10548 }

10549 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10550 {
10551     Unknown~cell.\\
10552     Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10553     the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10554     can't~be~executed~because~a~cell~doesn't~exist.\\
10555     This~command~ \token_to_str:N \line \ will~be~ignored.
10556 }

10557 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10558 {
10559     Duplicate~name.\\
10560     The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10561     in~this~ \@@_full_name_env: .\\
10562     This~key~will~be~ignored.\\
10563     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10564     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10565 }
10566 {
10567     The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
10568     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10569 }

```



```

10570 \@@_msg_new:nn { r-or-l-with-preamble }
10571 {
10572   Erroneous-use.\
10573   You-can't-use-the-key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
10574   You-must-specify-the-alignment-of~your~columns-with-the-preamble-of~
10575   your~ \@@_full_name_env: .\
10576   This-key-will-be-ignored.
10577 }
10578 \@@_msg_new:nn { Hdotsfor-in-col-0 }
10579 {
10580   Erroneous-use.\
10581   You-can't-use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\
10582   in~an~exterior~column~of~
10583   the~array.~This-error-is~fatal.
10584 }
10585 \@@_msg_new:nn { bad-corner }
10586 {
10587   Bad-corner.\
10588   #1-is~an~incorrect~specification~for~a~corner~(in~the~key~
10589   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\
10590   This-specification-of~corner~will~be~ignored.
10591 }
10592 \@@_msg_new:nn { bad-border }
10593 {
10594   Bad-border.\
10595   \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
10596   (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
10597   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10598   also~use~the~key~'tikz'
10599   \IfPackageLoadedF { tikz }
10600   { ~if~you~load~the~LaTeX~package~'tikz' } ).\
10601   This-specification-of~border~will~be~ignored.
10602 }
10603 \@@_msg_new:nn { TikzEveryCell-without-tikz }
10604 {
10605   TikZ-not-loaded.\
10606   You-can't-use~ \token_to_str:N \TikzEveryCell \
10607   because~you~have~not~loaded~tikz.~
10608   This-command-will-be-ignored.
10609 }
10610 \@@_msg_new:nn { tikz-key-without-tikz }
10611 {
10612   TikZ-not-loaded.\
10613   You-can't-use~the~key~'tikz'~for~the~command~' \token_to_str:N
10614   \Block '~because~you~have~not~loaded~tikz.~
10615   This-key-will-be-ignored.
10616 }
10617 \@@_msg_new:nn { Bad-argument-for-Block }
10618 {
10619   Bad-argument.\
10620   The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
10621   be~of~the~form~'i-j'~(or~completely~empty)~and~you~have~used:~
10622   '#1'. \
10623   If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono-cell~(as~if~
10624   the~argument~was~empty).
10625 }
10626 \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
10627 {
10628   Erroneous-use.\
10629   In~the~ \@@_full_name_env: ,~you~must~use~the~key~
10630   'last-col'~without~value.\

```

```

10631     However,~you~can~go~on~for~this~time~
10632     (the~value~' \l_keys_value_tl '~will~be~ignored).
10633 }

10634 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
10635 {
10636     Erroneous-use. \\
10637     In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
10638     'last-col'~without~value. \\
10639     However,~you~can~go~on~for~this~time~
10640     (the~value~' \l_keys_value_tl '~will~be~ignored).
10641 }

10642 \@@_msg_new:nn { Block-too-large-1 }
10643 {
10644     Block-too-large. \\
10645     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
10646     too~small~for~that~block. \\
10647     This~block~and~maybe~others~will~be~ignored.
10648 }

10649 \@@_msg_new:nn { Block-too-large-2 }
10650 {
10651     Block-too-large. \\
10652     The~preamble~of~your~ \@@_full_name_env: \ announces~ \int_use:N
10653     \g_@@_static_num_of_col_int \
10654     columns~but~you~use~only~ \int_use:N \c@jCol \ and~that's~why~a~block~
10655     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
10656     (&)~at~the~end~of~the~first~row~of~your~ \@@_full_name_env: . \\
10657     This~block~and~maybe~others~will~be~ignored.
10658 }

10659 \@@_msg_new:nn { unknown~column~type }
10660 {
10661     Bad~column~type. \\
10662     The~column~type~'#1'~in~your~ \@@_full_name_env: \
10663     is~unknown. \\
10664     This~error~is~fatal.
10665 }

10666 \@@_msg_new:nn { unknown~column~type~multicolumn }
10667 {
10668     Bad~column~type. \\
10669     The~column~type~'#1'~in~the~command~\token_to_str:N \multicolumn \
10670     ~of~your~ \@@_full_name_env: \
10671     is~unknown. \\
10672     This~error~is~fatal.
10673 }

10674 \@@_msg_new:nn { unknown~column~type~S }
10675 {
10676     Bad~column~type. \\
10677     The~column~type~'S'~in~your~ \@@_full_name_env: \ is~unknown. \\
10678     If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10679     load~that~package. \\
10680     This~error~is~fatal.
10681 }

10682 \@@_msg_new:nn { unknown~column~type~S~multicolumn }
10683 {
10684     Bad~column~type. \\
10685     The~column~type~'S'~in~the~command~\token_to_str:N \multicolumn \
10686     ~of~your~ \@@_full_name_env: \ is~unknown. \\
10687     If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10688     load~that~package. \\
10689     This~error~is~fatal.
10690 }

```

```

10691 \@@_msg_new:nn { tabularnote~forbidden }
10692 {
10693   Forbidden~command. \\
10694   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10695   ~here.~This~command~is~available~only~in~
10696   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
10697   the~argument~of~a~command~\token_to_str:N \caption \ included~
10698   in~an~environment~\{table\}. \\
10699   This~command~will~be~ignored.
10700 }
10701 \@@_msg_new:nn { borders~forbidden }
10702 {
10703   Forbidden~key.\\
10704   You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \
10705   because~the~option~'rounded~corners'~
10706   is~in~force~with~a~non~zero~value.\\
10707   This~key~will~be~ignored.
10708 }
10709 \@@_msg_new:nn { bottomrule~without~booktabs }
10710 {
10711   booktabs~not~loaded.\\
10712   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
10713   loaded~'booktabs'.\\
10714   This~key~will~be~ignored.
10715 }
10716 \@@_msg_new:nn { enumitem~not~loaded }
10717 {
10718   enumitem~not~loaded. \\
10719   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10720   ~because~you~haven't~loaded~'enumitem'. \\
10721   All~the~commands~ \token_to_str:N \tabularnote \ will~be~
10722   ignored~in~the~document.
10723 }
10724 \@@_msg_new:nn { tikz~without~tikz }
10725 {
10726   TikZ~not~loaded. \\
10727   You~can't~use~the~key~'tikz'~here~because~TikZ~is~not~
10728   loaded.~If~you~go~on,~that~key~will~be~ignored.
10729 }
10730 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
10731 {
10732   TikZ~not~loaded. \\
10733   You~have~used~the~key~'tikz'~in~the~definition~of~a~
10734   customized~line~(with~'custom~line')~but~TikZ~is~not~loaded.~
10735   You~can~go~on~but~you~will~have~another~error~if~you~actually~
10736   use~that~custom~line.
10737 }
10738 \@@_msg_new:nn { tikz~in~borders~without~tikz }
10739 {
10740   TikZ~not~loaded. \\
10741   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
10742   command~' \token_to_str:N \Block ')~but~TikZ~is~not~loaded.~
10743   That~key~will~be~ignored.
10744 }
10745 \@@_msg_new:nn { color~in~custom~line~with~tikz }
10746 {
10747   Erroneous~use.\\
10748   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
10749   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
10750   The~key~'color'~will~be~discarded.
10751 }

```

```

10752 \@@_msg_new:nn { Wrong-last-row }
10753 {
10754   Wrong-number.\
10755   You-have-used-'last-row= \int_use:N \l_@@_last_row_int '~but-your~
10756   \@@_full_name_env: \ seems-to-have~ \int_use:N \c@iRow \ rows.~
10757   If-you-go-on,~the-value-of~ \int_use:N \c@iRow \ will-be-used-for~
10758   last-row-but-you-should-correct-your-code.~You-can-avoid-this~
10759   problem-by-using-'last-row'~without-value~(more-compilations~
10760   might-be-necessary).
10761 }
10762 \@@_msg_new:nn { Yet-in-env }
10763 {
10764   Nested-environments.\
10765   Environments-of~nicematrix~can't-be-nested.\
10766   This-error-is-fatal.
10767 }
10768 \@@_msg_new:nn { Outside-math-mode }
10769 {
10770   Outside-math-mode.\
10771   The-\@@_full_name_env: \ can-be-used-only-in-math-mode~
10772   (and-not-in~ \token_to_str:N \vcenter ).\
10773   This-error-is-fatal.
10774 }
10775 \@@_msg_new:nn { One-letter-allowed }
10776 {
10777   Bad-name.\
10778   The-value-of-key~' \l_keys_key_str '~must-be-of-length~1-and~
10779   you-have-used~' \l_keys_value_tl '~.\
10780   It-will-be-ignored.
10781 }
10782 \@@_msg_new:nn { TabularNote-in-CodeAfter }
10783 {
10784   Environment~\{TabularNote\}~forbidden.\
10785   You-must-use~\{TabularNote\}~at-the-end-of-your~\{NiceTabular\}~
10786   but~*before*~the~ \token_to_str:N \CodeAfter . \
10787   This-environment~\{TabularNote\}~will-be-ignored.
10788 }
10789 \@@_msg_new:nn { varwidth-not-loaded }
10790 {
10791   varwidth-not-loaded.\
10792   You-can't-use-the-column-type~'V'~because~'varwidth'~is-not~
10793   loaded.\
10794   Your~column~will-behave-like~'p'.
10795 }
10796 \@@_msg_new:nn { varwidth-not-loaded-in-X }
10797 {
10798   varwidth-not-loaded.\
10799   You-can't-use-the-key~'V'~in-your~column~'X'~
10800   because~'varwidth'~is-not-loaded.\
10801   It-will-be-ignored. \
10802 }
10803 \@@_msg_new:nnn { Unknown-key-for-RulesBis }
10804 {
10805   Unknown-key.\
10806   Your-key~' \l_keys_key_str '~is-unknown-for-a-rule.\
10807   \c_@@_available_keys_str
10808 }
10809 {
10810   The-available-keys-are~(in-alphabetic-order):~
10811   color,~
10812   dotted,~

```

```

10813     multiplicity,~
10814     sep-color,~
10815     tikz,~and~total-width.
10816 }
10817
10818 \@@_msg_new:nnn { Unknown~key~for~Block }
10819 {
10820     Unknown~key. \\
10821     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10822     \token_to_str:N \Block . \\
10823     It~will~be~ignored. \\
10824     \c_@@_available_keys_str
10825 }
10826 {
10827     The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10828     b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
10829     opacity,~rounded-corners,~r,~respect~arraystretch,~t,~T,~tikz,~transparent~
10830     and~vlines.
10831 }
10832 \@@_msg_new:nnn { Unknown~key~for~Brace }
10833 {
10834     Unknown~key. \\
10835     The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
10836     \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
10837     It~will~be~ignored. \\
10838     \c_@@_available_keys_str
10839 }
10840 {
10841     The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
10842     right~shorten,~shorten~(which~fixes~both~left~shorten~and~
10843     right~shorten)~and~yshift.
10844 }
10845 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10846 {
10847     Unknown~key. \\
10848     The~key~' \l_keys_key_str '~is~unknown. \\
10849     It~will~be~ignored. \\
10850     \c_@@_available_keys_str
10851 }
10852 {
10853     The~available~keys~are~(in~alphabetic~order):~
10854     delimiters/color,~
10855     rules~(with~the~subkeys~'color'~and~'width'),~
10856     sub-matrix~(several~subkeys)~
10857     and~xdots~(several~subkeys).~
10858     The~latter~is~for~the~command~ \token_to_str:N \line .
10859 }
10860 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10861 {
10862     Unknown~key. \\
10863     The~key~' \l_keys_key_str '~is~unknown. \\
10864     It~will~be~ignored. \\
10865     \c_@@_available_keys_str
10866 }
10867 {
10868     The~available~keys~are~(in~alphabetic~order):~
10869     create-cell-nodes,~
10870     delimiters/color~and~
10871     sub-matrix~(several~subkeys).
10872 }
10873 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10874 {

```

```

10875     Unknown~key.\\
10876     The~key~' \l_keys_key_str '~is~unknown.\\
10877     That~key~will~be~ignored. \\
10878     \c_@@_available_keys_str
10879 }
10880 {
10881     The~available~keys~are~(in~alphabetic~order):~
10882     'delimiters/color',~
10883     'extra-height',~
10884     'hlines',~
10885     'hvlines',~
10886     'left-xshift',~
10887     'name',~
10888     'right-xshift',~
10889     'rules'~(with~the~subkeys~'color'~and~'width'),~
10890     'slim',~
10891     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10892     and~'right-xshift').\\
10893 }
10894 \@@_msg_new:nnn { Unknown~key~for~notes }
10895 {
10896     Unknown~key.\\
10897     The~key~' \l_keys_key_str '~is~unknown.\\
10898     That~key~will~be~ignored. \\
10899     \c_@@_available_keys_str
10900 }
10901 {
10902     The~available~keys~are~(in~alphabetic~order):~
10903     bottomrule,~
10904     code-after,~
10905     code-before(+),~
10906     detect-duplicates,~
10907     enumitem-keys,~
10908     enumitem-keys-para,~
10909     para,~
10910     label-in-list,~
10911     label-in-tabular~and~
10912     style.
10913 }
10914 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10915 {
10916     Unknown~key.\\
10917     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10918     \token_to_str:N \RowStyle . \\
10919     That~key~will~be~ignored. \\
10920     \c_@@_available_keys_str
10921 }
10922 {
10923     The~available~keys~are~(in~alphabetic~order):~
10924     bold,~
10925     cell-space-top-limit(+),~
10926     cell-space-bottom-limit(+),~
10927     cell-space-limits(+),~
10928     color,~
10929     fill~(alias:~rowcolor),~
10930     nb-rows,~
10931     opacity~and~
10932     rounded-corners.
10933 }
10934 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10935 {
10936     Unknown~key.\\
10937     The~key~' \l_keys_key_str '~is~unknown~for~the~command~

```

```

10938 \token_to_str:N \NiceMatrixOptions . \\
10939 That~key~will~be~ignored. \\
10940 \c_@@_available_keys_str
10941 }
10942 {
10943   The~available~keys~are~(in~alphabetic~order):~
10944   &-in-blocks,~
10945   allow-duplicate-names,~
10946   ampersand-in-blocks,~
10947   caption-above,~
10948   cell-space-bottom-limit(+),~
10949   cell-space-limits(+),~
10950   cell-space-top-limit(+),~
10951   code-for-first-col(+),~
10952   code-for-first-row(+),~
10953   code-for-last-col(+),~
10954   code-for-last-row(+),~
10955   corners,~
10956   custom-key,~
10957   create-extra-nodes,~
10958   create-medium-nodes,~
10959   create-large-nodes,~
10960   custom-line,~
10961   delimiters~(several~subkeys),~
10962   end-of-row,~
10963   first-col,~
10964   first-row,~
10965   hlines,~
10966   hvlines,~
10967   hvlines-except-borders,~
10968   last-col,~
10969   last-row,~
10970   left-margin,~
10971   light-syntax,~
10972   light-syntax-expanded,~
10973   matrix/columns-type,~
10974   no-cell-nodes,~
10975   notes~(several~subkeys),~
10976   nullify-dots,~
10977   pgf-node-code,~
10978   renew-dots,~
10979   renew-matrix,~
10980   respect-arraystretch,~
10981   rounded-corners,~
10982   right-margin,~
10983   rules~(with~the~subkeys~'color'~and~'width'),~
10984   small,~
10985   sub-matrix~(several~subkeys),~
10986   vlines,~
10987   xdots~(several~subkeys).
10988 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

10989 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10990 {
10991   Unknown~key.\\
10992   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
10993   \{NiceArray\}. \\
10994   That~key~will~be~ignored. \\
10995   \c_@@_available_keys_str
10996 }
10997 {
10998   The~available~keys~are~(in~alphabetic~order):~

```

```

11099 &-in-blocks,~
11000 ampersand-in-blocks,~
11001 b,~
11002 baseline,~
11003 c,~
11004 cell-space-bottom-limit,~
11005 cell-space-limits,~
11006 cell-space-top-limit,~
11007 code-after,~
11008 code-for-first-col(+),~
11009 code-for-first-row(+),~
11010 code-for-last-col(+),~
11011 code-for-last-row(+),~
11012 columns-width,~
11013 corners,~
11014 create-extra-nodes,~
11015 create-medium-nodes,~
11016 create-large-nodes,~
11017 extra-left-margin,~
11018 extra-right-margin,~
11019 first-col,~
11020 first-row,~
11021 hlines,~
11022 hvlines,~
11023 hvlines-except-borders,~
11024 last-col,~
11025 last-row,~
11026 left-margin,~
11027 light-syntax,~
11028 light-syntax-expanded,~
11029 name,~
11030 no-cell-nodes,~
11031 nullify-dots,~
11032 pgf-node-code,~
11033 renew-dots,~
11034 respect-arraystretch,~
11035 right-margin,~
11036 rounded-corners,~
11037 rules~(with~the~subkeys~'color'~and~'width'),~
11038 small,~
11039 t,~
11040 vlides,~
11041 xdots/color,~
11042 xdots/shorten-start(+),~
11043 xdots/shorten-end(+),~
11044 xdots/shorten(+)-and~
11045 xdots/line-style.
11046 }

```

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`).

```

11047 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
11048 {
11049   Unknown~key.\\
11050   The~key~' \l_keys_key_str '~is~unknown~for~the~
11051   \@@_full_name_env: . \\
11052   That~key~will~be~ignored. \\
11053   \c_@@_available_keys_str
11054 }
11055 {
11056   The~available~keys~are~(in~alphabetic~order):~
11057   &-in-blocks,~
11058   ampersand-in-blocks,~

```



```

11059 b,~
11060 baseline,~
11061 c,~
11062 cell-space-bottom-limit,~
11063 cell-space-limits,~
11064 cell-space-top-limit,~
11065 code-after,~
11066 code-for-first-col(+),~
11067 code-for-first-row(+),~
11068 code-for-last-col(+),~
11069 code-for-last-row(+),~
11070 columns-type,~
11071 columns-width,~
11072 corners,~
11073 create-extra-nodes,~
11074 create-medium-nodes,~
11075 create-large-nodes,~
11076 extra-left-margin,~
11077 extra-right-margin,~
11078 first-col,~
11079 first-row,~
11080 hlines,~
11081 hvlines,~
11082 hvlines-except-borders,~
11083 l,~
11084 last-col,~
11085 last-row,~
11086 left-margin,~
11087 light-syntax,~
11088 light-syntax-expanded,~
11089 name,~
11090 no-cell-nodes,~
11091 nullify-dots,~
11092 pgf-node-code,~
11093 r,~
11094 renew-dots,~
11095 respect-arraystretch,~
11096 right-margin,~
11097 rounded-corners,~
11098 rules~(with~the~subkeys~'color'~and~'width'),~
11099 small,~
11100 t,~
11101 vlines,~
11102 xdots/color,~
11103 xdots/shorten-start(+),~
11104 xdots/shorten-end(+),~
11105 xdots/shorten(+),~and~
11106 xdots/line-style.
11107 }

11108 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
11109 {
11110   Unknown~key.\\
11111   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11112   \{NiceTabular\}. \\
11113   That~key~will~be~ignored. \\
11114   \c_@@_available_keys_str
11115 }
11116 {
11117   The~available~keys~are~(in~alphabetic~order):~
11118   &~in~blocks,~
11119   ampersand~in~blocks,~
11120   b,~
11121   baseline,~

```

```

11122 c,~
11123 caption,~
11124 cell-space-bottom-limit,~
11125 cell-space-limits,~
11126 cell-space-top-limit,~
11127 code-after,~
11128 code-for-first-col(+),~
11129 code-for-first-row(+),~
11130 code-for-last-col(+),~
11131 code-for-last-row(+),~
11132 columns-width,~
11133 corners,~
11134 custom-line,~
11135 create-extra-nodes,~
11136 create-medium-nodes,~
11137 create-large-nodes,~
11138 extra-left-margin,~
11139 extra-right-margin,~
11140 first-col,~
11141 first-row,~
11142 hlines,~
11143 hvlines,~
11144 hvlines-except-borders,~
11145 label,~
11146 last-col,~
11147 last-row,~
11148 left-margin,~
11149 light-syntax,~
11150 light-syntax-expanded,~
11151 name,~
11152 no-cell-nodes,~
11153 notes~(several~subkeys),~
11154 nullify-dots,~
11155 pgf-node-code,~
11156 renew-dots,~
11157 respect-arraystretch,~
11158 right-margin,~
11159 rounded-corners,~
11160 rules~(with~the~subkeys~'color'~and~'width'),~
11161 short-caption,~
11162 t,~
11163 tabularnote,~
11164 vlines,~
11165 xdots/color,~
11166 xdots/shorten-start(+),~
11167 xdots/shorten-end(+),~
11168 xdots/shorten(+),~and~
11169 xdots/line-style.
11170 }

11171 \@@_msg_new:nnn { Duplicate-name }
11172 {
11173   Duplicate-name.\
11174   The-name-' \l_keys_value_tl 'is-already-used-and-you-shouldn't-use~
11175   the-same-environment-name-twice.~You-can-go-on,~but,~
11176   maybe,~you-will-have-incorrect-results-especially~
11177   if-you-use~'columns-width=auto'.~If-you-don't-want-to-see-this~
11178   message-again,~use-the-key~'allow-duplicate-names'~in~
11179   ' \token_to_str:N \NiceMatrixOptions '\
11180   \bool_if:NF \g_@@_messages_for_Overleaf_bool
11181   { For-a-list-of-the-names-already-used,~type-H~<return>. }
11182 }
11183 {
11184   The-names-already-defined-in-this-document-are:-

```

```

11185 \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
11186 }
11187 \@@_msg_new:nn { caption-above-in-env }
11188 {
11189   The-key~'caption-above'~must~be~used~in~\token_to_str:N \NiceMatrixOptions.\\
11190   That~key~will~be~ignored.
11191 }
11192 \@@_msg_new:nn { show-cell-names }
11193 {
11194   There-is-no-key~'show-cell-names'~in~nicematrix.\\
11195   You-should-use-the-command~\token_to_str:N \ShowCellNames\
11196   in-the~\token_to_str:N \CodeBefore\ or-the~\token_to_str:N
11197   \CodeAfter. \\
11198   That~key~will~be~ignored.
11199 }
11200 \@@_msg_new:nn { Option-auto-for-columns-width }
11201 {
11202   Erroneous~use.\\
11203   You-can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
11204   That~key~will~be~ignored.
11205 }
11206 \@@_msg_new:nn { NiceTabularX~without~X }
11207 {
11208   NiceTabularX~without~X.\\
11209   You-should-not-use-~\{NiceTabularX\}~without~X~columns.\\
11210   However,~you~can~go~on.
11211 }
11212 \@@_msg_new:nn { Preamble-forgotten }
11213 {
11214   Preamble-forgotten.\\
11215   You-have-probably~forgotten~the~preamble~of~your~
11216   \@@_full_name_env: . \\
11217   This~error~is~fatal.
11218 }
11219 \@@_msg_new:nn { Invalid-col-number }
11220 {
11221   Invalid~column~number.\\
11222   A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11223   specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.
11224 }
11225 \@@_msg_new:nn { Invalid-row-number }
11226 {
11227   Invalid~row~number.\\
11228   A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11229   specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.
11230 }
11231 \@@_define_com:NNN p ( )
11232 \@@_define_com:NNN b [ ]
11233 \@@_define_com:NNN v | |
11234 \@@_define_com:NNN V \l \l
11235 \@@_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	63
12	The redefinition of <code>\multicolumn</code>	80
13	The environment <code>{NiceMatrix}</code> and its variants	97
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	98
15	After the construction of the array	100
16	We draw the dotted lines	106
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-after	136
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	170
24	The environment <code>{NiceMatrixBlock}</code>	172
25	The extra nodes	174
26	The blocks	178
27	How to draw the dotted lines transparently	205
28	Automatic arrays	205
29	The redefinition of the command <code>\dotfill</code>	207
30	The command <code>\diagbox</code>	207

31	The keyword <code>\CodeAfter</code>	208
32	The delimiters in the preamble	209
33	The command <code>\SubMatrix</code>	210
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	219
35	The commands <code>HBrace</code> et <code>VBrace</code>	222
36	The command <code>TikzEveryCell</code>	225
37	The command <code>\CreateBlocksInColumn</code>	227
38	The command <code>\ShowCellNames</code>	227
39	We process the options at package loading	229
40	About the package underscore	231
41	Error messages of the package	231