

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

September 30, 2025

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

```
3 \ProvidesExplPackage
4   {nicematrix}
5   {\myfiledate}
6   {\myfileversion}
7   {Enhanced arrays with the help of PGF/TikZ}
8 \msg_new:nnn { nicematrix } { latex-too-old }
9   {
10    Your~LaTeX~release~is~too~old. \\
11    You~need~at~least~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 7.3 of `nicematrix`, at the date of 2025/09/30.

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 }

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

```

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

```

82 \tl_new:N \l_@@_argspec_tl

```

```

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

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

```

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.

```

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

```

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`).

```

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

```

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

```

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

```

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.

```

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

```

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

```

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

```

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.

```

139 \hook_gput_code:nnn { begindocument } { . }
140 {
141   \IfPackageLoadedT { booktabs }
142   { \iow_now:Nn \@mainaux { \nicematrix@redefine@check@rerun } }
143 }
144 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
145 {
146   \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`).

```

147   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
148   {
149     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } { 1 } { 3 } }
150     { \@@_old_pgfutil@check@rerun { ##1 } { ##2 } }
151   }
152 }

```

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

```

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

```

```

170     }
171 }

```

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).

```

172 \cs_new_protected:Npn \@@_replace_columncolor:
173 {
174     \tl_replace_all:Nnn \g_@@_array_preamble_tl
175     { \columncolor }
176     { \@@_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.

```

177 }
178 }
179 {
180     \cs_new_protected:Npn \@@_revert_colortbl: { }
181     \cs_new_protected:Npn \@@_replace_columncolor:
182     { \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.

```

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

```

Idem for `\CT@drs@`.

```

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

```

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.

```

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

```

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

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

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`.

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

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.

```
224 \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`.

```
225 { \@@_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*.

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

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).

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

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

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

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

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

¹See question 99041 on TeX StackExchange.

```

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

```

```

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

```

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

```

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 \dim_new:N \l_@@_tmpe_dim
291 \dim_new:N \l_@@_tmpf_dim

292 \tl_new:N \l_@@_tmpc_tl
293 \tl_new:N \l_@@_tmpd_tl

294 \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.

```
295 \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).

```
296 \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.

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

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

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

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

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

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

```
302 \bool_new:N \g_@@_delims_bool
303 \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.

```
304 \bool_new:N \l_@@_preamble_bool
305 \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.

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

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

```
307 \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.

```
308 \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).

```
309 \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.).

```
310 \dim_new:N \l_@@_col_width_dim
311 \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.

```
312 \int_new:N \g_@@_row_total_int
313 \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).

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

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

```
315 \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.

```
316 \tl_new:N \l_@@_hpos_cell_tl
317 \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`.

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

Idem for the mono-row blocks.

```
319 \dim_new:N \g_@@_blocks_ht_dim
320 \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}`).

```
321 \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`.

```
322 \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.

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

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

```
324 \bool_new:N \l_@@_notes_detect_duplicates_bool
325 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

```
326 \bool_new:N \l_@@_initial_open_bool
327 \bool_new:N \l_@@_final_open_bool
328 \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.

```
329 \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).

```
330 \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).

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

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

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

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

```
333 \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.

```
334 \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`).

```
335 \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`.

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

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

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

```
338 \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 }`).

```
339 \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.

```
340 \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.

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

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

```
343 \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}`).

```
344 \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`).

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

For `\multicolumn`.

```
346 \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`.

```
347 \tl_new:N \l_@@_columns_type_tl
348 \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 `:`.

```
349 \tl_new:N \l_@@_xdots_down_tl
350 \tl_new:N \l_@@_xdots_up_tl
351 \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`).

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

```
353 \cs_new_protected:Npn \@@_test_if_math_mode:
354 {
355   \if_mode_math: \else:
356     \@@_fatal:n { Outside~math~mode }
357   \fi:
358 }
```

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

```
359 \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”.

```
360 \colorlet { nicematrix-last-col } { . }
361 \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`).

```
362 \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*.

```
363 \tl_new:N \g_@@_com_or_env_str
364 \tl_gset:Nn \g_@@_com_or_env_str { environment }
```

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

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

```
366 \cs_new:Npn \@@_full_name_env:
367 {
368   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
369     { command \space \c_backslash_str \g_@@_name_env_str }
370     { environment \space \{ \g_@@_name_env_str \} }
371 }
```

```
372 \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.

```
373 \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.

```
374 \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.

```
375 \tl_new:N \g_@@_pre_code_before_tl
376 \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.

```
377 \tl_new:N \g_@@_pre_code_after_tl
378 \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.

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

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

```
380 \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.

```
381 \int_new:N \l_@@_old_iRow_int
382 \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).

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

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

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

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

```
385 \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.

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

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

```
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
```

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

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

```
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
```

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

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the Δ_x and Δ_y of the first `\Ddots` diagonal. We have to store these values in order to draw the others `\Ddots` diagonals parallel to the first one. Similarly `\g_@@_delta_x_two_dim` and `\g_@@_delta_y_two_dim` are the Δ_x and Δ_y of the first `\Iddots` diagonal.

```

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

```

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
```

- **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:

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

```

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

```

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

```

478 \bool_new:N \g_@@_last_col_found_bool

```

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

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

```

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.

```

485 \cs_new_protected:Npn \@@_expand_clist:N #1
486 {
487 \clist_if_in:NnF #1 { all }
488 {
489 \clist_clear:N \l_tmpa_clist
490 \clist_map_inline:Nn #1
491 {

```

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

```

492 \tl_if_in:nnTF { ##1 } { - }
493 { \@@_cut_on_hyphen:w ##1 \q_stop }
494 {

```

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

```

495 \def \l_tmpa_tl { ##1 }
496 \def \l_tmpb_tl { ##1 }
497 }
498 \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
499 { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
500 }
501 \tl_set_eq:NN #1 \l_tmpa_clist
502 }
503 }

```

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.

```

504 \hook_gput_code:nnn { begindocument } { . }
505 {
506 \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
507 \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
508 }

```

5 The command `\tabularnote`

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

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

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

```
509 \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`.

```
510 \int_new:N \g_@@_tabularnote_int
511 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }
512 \seq_new:N \g_@@_notes_seq
513 \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.

```
514 \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.

```
515 \seq_new:N \l_@@_notes_labels_seq
516 \newcounter { nicematrix_draft }
```

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

```

517 \cs_new_protected:Npn \@@_notes_format:n #1
518 {
519   \setcounter { nicematrix_draft } { #1 }
520   \@@_notes_style:n { nicematrix_draft }
521 }

```

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

```

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

```

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

```

523 \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`.

```

524 \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`.

```

525 \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`).

```

526 \hook_gput_code:nnn { begindocument } { . }
527 {
528   \IfPackageLoadedTF { enumitem }
529   {

```

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.

```

530     \newlist { tabularnotes } { enumerate } { 1 }
531     \setlist [ tabularnotes ]
532     {
533       topsep = \c_zero_dim ,
534       noitemsep ,
535       leftmargin = * ,
536       align = left ,
537       labelsep = \c_zero_dim ,
538       label =
539         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
540     }
541     \newlist { tabularnotes* } { enumerate* } { 1 }
542     \setlist [ tabularnotes* ]
543     {
544       afterlabel = \nobreak ,
545       itemjoin = \quad ,
546       label =
547         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
548     }

```

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`.

```

549     \NewDocumentCommand \tabularnote { o m }
550     {
551       \bool_lazy_or:nnT { \cs_if_exist_p:N \capttype } { \l_@@_in_env_bool }

```

```

552     {
553         \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } { \l_@@_in_env_bool }
554         { \@@_error:n { tabularnote~forbidden } }
555         {
556             \bool_if:NTF \l_@@_in_caption_bool
557             \@@_tabularnote_caption:nn
558             \@@_tabularnote:nn
559             { #1 } { #2 }
560         }
561     }
562 }
563 }
564 {
565     \NewDocumentCommand \tabularnote { o m }
566     { \@@_err_enumitem_not_loaded: }
567 }
568 }
569 \cs_new_protected:Npn \@@_err_enumitem_not_loaded:
570 {
571     \@@_error_or_warning:n { enumitem~not~loaded }
572     \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
573 }
574 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
575 { \tl_if_novalue:nT { #1 } { #3 } }

```

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

```

576 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
577 {

```

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).

```

578     \int_zero:N \l_tmpa_int
579     \bool_if:NTF \l_@@_notes_detect_duplicates_bool
580     {

```

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

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

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

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

```

581     \int_zero:N \l_tmpb_int
582     \seq_map_indexed_inline:Nn \g_@@_notes_seq
583     {
584         \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
585         \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
586         {
587             \tl_if_novalue:nTF { #1 }
588             { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
589             { \int_set:Nn \l_tmpa_int { ##1 } }
590             \seq_map_break:
591         }
592     }
593     \int_if_zero:nF { \l_tmpa_int }
594     { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }

```

```

595     }
596     \int_if_zero:nT { \l_tmpa_int }
597     {
598         \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
599         \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
600     }
601     \seq_put_right:Ne \l_@@_notes_labels_seq
602     {
603         \tl_if_novalue:nTF { #1 }
604         {
605             \@@_notes_format:n
606             {
607                 \int_eval:n
608                 {
609                     \int_if_zero:nTF { \l_tmpa_int }
610                     { \c@tabularnote }
611                     { \l_tmpa_int }
612                 }
613             }
614         }
615         { #1 }
616     }
617     \peek_meaning:NF \tabularnote
618     {

```

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`.

```

619     \hbox_set:Nn \l_tmpa_box
620     {

```

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

```

621     \@@_notes_label_in_tabular:n
622     {
623         \seq_use:Nnnn
624         \l_@@_notes_labels_seq { , } { , } { , }
625     }
626 }

```

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

```

627     \int_gdecr:N \c@tabularnote
628     \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.

```

629     \int_gincr:N \g_@@_tabularnote_int
630     \refstepcounter { tabularnote }
631     \int_compare:nNnT { \l_tmpa_int } = { \c@tabularnote }
632     { \int_gincr:N \c@tabularnote }
633     \seq_clear:N \l_@@_notes_labels_seq
634     \bool_lazy_or:nnTF
635     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { c } }
636     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { r } }
637     {
638         \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).

```

639         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
640     }
641     { \box_use:N \l_tmpa_box }
642 }

```

```
643 }
```

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`.

```
644 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
645 {
646   \bool_if:NTF \g_@@_caption_finished_bool
647   {
648     \int_compare:nNnT { \c@tabularnote } = { \g_@@_notes_caption_int }
649     { \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`!

```
650     \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
651     { \@@_error:n { Identical~notes~in~caption } }
652   }
653 }
```

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

```
654     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
655     {
```

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.

```
656       \bool_gset_true:N \g_@@_caption_finished_bool
657       \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
658       \int_gzero:N \c@tabularnote
659     }
660     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
661 }
```

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!

```
662   \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
663   \seq_put_right:Ne \l_@@_notes_labels_seq
664   {
665     \tl_if_novalue:nTF { #1 }
666     { \@@_notes_format:n { \int_use:N \c@tabularnote } }
667     { #1 }
668   }
669   \peek_meaning:NF \tabularnote
670   {
671     \@@_notes_label_in_tabular:n
672     { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
673     \seq_clear:N \l_@@_notes_labels_seq
674   }
675 }

676 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
677 { \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.

```

678 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
679 {
680   \begin { pgfscope }
681   \pgfset
682   {
683     inner~sep = \c_zero_dim ,
684     minimum~size = \c_zero_dim
685   }
686   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
687   \pgfnode
688   { rectangle }
689   { center }
690   {
691     \vbox_to_ht:nn
692     { \dim_abs:n { #5 - #3 } }
693     {
694       \vfill
695       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
696     }
697   }
698   { #1 }
699   { }
700   \end { pgfscope }
701 }

```

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.

```

702 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
703 {
704   \begin { pgfscope }
705   \pgfset
706   {
707     inner~sep = \c_zero_dim ,
708     minimum~size = \c_zero_dim
709   }
710   \pgftransformshift { \pgfpoint scale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
711   \pgfpointdiff { #3 } { #2 }
712   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
713   \pgfnode
714   { rectangle }
715   { center }
716   {
717     \vbox_to_ht:nn
718     { \dim_abs:n \l_tmpb_dim }
719     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
720   }
721   { #1 }
722   { }
723   \end { pgfscope }
724 }

```

7 The options

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

```

725 \tl_new:N \l_@@_caption_tl

```

```

726 \tl_new:N \l_@@_short_caption_tl
727 \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).

```

728 \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`.

```

729 \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`).

```

730 \dim_new:N \l_@@_cell_space_top_limit_dim
731 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

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

```

732 \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.

```

733 \dim_new:N \l_@@_xdots_inter_dim
734 \hook_gput_code:nnn { begindocument } { . }
735 { \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).

```

736 \dim_new:N \l_@@_xdots_shorten_start_dim
737 \dim_new:N \l_@@_xdots_shorten_end_dim
738 \hook_gput_code:nnn { begindocument } { . }
739 {
740   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
741   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
742 }

```

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.

```

743 \dim_new:N \l_@@_xdots_radius_dim
744 \hook_gput_code:nnn { begindocument } { . }
745 { \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.

```

746 \tl_new:N \l_@@_xdots_line_style_tl
747 \tl_const:Nn \c_@@_standard_tl { standard }
748 \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`.

```
749 \bool_new:N \l_@@_light_syntax_bool
750 \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).

```
751 \tl_new:N \l_@@_baseline_tl
752 \tl_set:Nn \l_@@_baseline_tl { c }
```

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

```
753 \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`).

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

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

```
755 \bool_new:N \l_@@_parallelize_diags_bool
756 \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.

```
757 \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.).

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

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

```
759 \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).

```
760 \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!

```
761 \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.

```
762 \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”.

```
763 \bool_new:N \l_@@_medium_nodes_bool
764 \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).

```
765 \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`).

```
766 \dim_new:N \l_@@_left_margin_dim
767 \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`.

```
768 \dim_new:N \l_@@_extra_left_margin_dim
769 \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.

```
770 \tl_new:N \l_@@_end_of_row_tl
771 \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 “:”.

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

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

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

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

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

```
775 \keys_define:nn { nicematrix / xdots }
776 {
777   Vbrace .bool_set:N = \l_@@_Vbrace_bool ,
778   shorten-start .code:n =
779     \hook_gput_code:nnn { begindocument } { . }
780     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
781   shorten-end .code:n =
782     \hook_gput_code:nnn { begindocument } { . }
783     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
784   shorten-start .value_required:n = true ,
785   shorten-end .value_required:n = true ,
786   shorten .code:n =
787     \hook_gput_code:nnn { begindocument } { . }
788     {
789       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
790       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
791     } ,
792   shorten .value_required:n = true ,
793   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
794   horizontal-labels .default:n = true ,
795   horizontal-label .bool_set:N = \l_@@_xdots_h_labels_bool ,
796   horizontal-label .default:n = true ,
797   line-style .code:n =
```

```

798 {
799   \bool_lazy_or:nnTF
800   { \cs_if_exist_p:N \tikzpicture }
801   { \str_if_eq_p:nn { #1 } { standard } }
802   { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
803   { \@@_error:n { bad-option-for-line-style } }
804 } ,
805 line-style .value_required:n = true ,
806 color .tl_set:N = \l_@@_xdots_color_tl ,
807 color .value_required:n = true ,
808 radius .code:n =
809   \hook_gput_code:nnn { begindocument } { . }
810   { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
811 radius .value_required:n = true ,
812 inter .code:n =
813   \hook_gput_code:nnn { begindocument } { . }
814   { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
815 radius .value_required:n = true ,

```

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

```

816   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
817   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
818   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).

```

819   draw-first .code:n = \prg_do_nothing: ,
820   unknown .code:n = \@@_error:n { Unknown~key~for~xdots }
821 }

```

```

822 \keys_define:nn { nicematrix / rules }
823 {
824   color .tl_set:N = \l_@@_rules_color_tl ,
825   color .value_required:n = true ,
826   width .dim_set:N = \arrayrulewidth ,
827   width .value_required:n = true ,
828   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
829 }
830 \cs_new_protected:Npn \@@_err_key_color_inside:
831 {
832   \@@_error_or_warning:n { key-color-inside }
833   \cs_gset:Npn \@@_err_key_color_inside: { }
834 }

```

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

```

835 \keys_define:nn { nicematrix / Global }
836 {
837   color-inside .code:n = \@@_err_key_color_inside: ,
838   colortbl-like .code:n = \@@_err_key_color_inside: ,
839   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
840   ampersand-in-blocks .default:n = true ,
841   &-in-blocks .meta:n = ampersand-in-blocks ,
842   no-cell-nodes .code:n =
843     \bool_set_true:N \l_@@_no_cell_nodes_bool
844     \cs_set_protected:Npn \@@_node_cell:
845       { \set@color \box_use_drop:N \l_@@_cell_box } ,
846   no-cell-nodes .value_forbidden:n = true ,
847   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,

```

```

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

```

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

```

897     { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
898 }
899 { \@@_error:n { One~letter~allowed } }
900 } ,
901 vlines-in-sub-matrix .value_required:n = true ,
902 hvlines .code:n =
903 {
904     \bool_set_true:N \l_@@_hvlines_bool
905     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
906     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
907 } ,
908 hvlines .value_forbidden:n = true ,
909 hvlines-except-borders .code:n =

```

```

910     {
911       \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
912       \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
913       \bool_set_true:N \l_@@_hvlines_bool
914       \bool_set_true:N \l_@@_except_borders_bool
915     } ,
916     hvlines-except-borders .value_forbidden:n = true ,
917     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.

```

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

```

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

```

942 \keys_define:nn { nicematrix / environments }
943 {
944   corners .clist_set:N = \l_@@_corners_clist ,
945   corners .default:n = { NW , SW , NE , SE } ,
946   code-before .code:n =
947     {
948       \tl_if_empty:nF { #1 }
949       {
950         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
951         \bool_set_true:N \l_@@_code_before_bool
952       }
953     } ,
954   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}`.

```

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

```

961     \str_if_eq:eeTF { #1 } { auto }
962     { \bool_set_true:N \l_@@_auto_columns_width_bool }
963     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
964     columns-width .value_required:n = true ,
965     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.

```

966     \legacy_if:nF { measuring@ }
967     {
968         \str_set:Ne \l_@@_name_str { #1 }
969         \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
970         { \@@_err_duplicate_names:n { #1 } }
971         { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
972     } ,
973     name .value_required:n = true ,
974     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
975     code-after .value_required:n = true ,
976 }

977 \cs_set:Npn \@@_err_duplicate_names:n #1
978 { \@@_error:nn { Duplicate~name } { #1 } }

979 \keys_define:nn { nicematrix / notes }
980 {
981     para .bool_set:N = \l_@@_notes_para_bool ,
982     para .default:n = true ,
983     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
984     code-before .value_required:n = true ,
985     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
986     code-after .value_required:n = true ,
987     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
988     bottomrule .default:n = true ,
989     style .cs_set:Np = \@@_notes_style:n #1 ,
990     style .value_required:n = true ,
991     label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
992     label-in-tabular .value_required:n = true ,
993     label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
994     label-in-list .value_required:n = true ,
995     enumitem-keys .code:n =
996     {
997         \hook_gput_code:nnn { begindocument } { . }
998         {
999             \IfPackageLoadedT { enumitem }
1000             { \setlist* [ tabularnotes ] { #1 } }
1001         }
1002     } ,
1003     enumitem-keys .value_required:n = true ,
1004     enumitem-keys-para .code:n =
1005     {
1006         \hook_gput_code:nnn { begindocument } { . }
1007         {
1008             \IfPackageLoadedT { enumitem }
1009             { \setlist* [ tabularnotes* ] { #1 } }
1010         }
1011     } ,
1012     enumitem-keys-para .value_required:n = true ,
1013     detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1014     detect-duplicates .default:n = true ,
1015     unknown .code:n = \@@_error:n { Unknown~key~for~notes }
1016 }

1017 \keys_define:nn { nicematrix / delimiters }

```



```

1018 {
1019   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1020   max-width .default:n = true ,
1021   color .tl_set:N = \l_@@_delimiters_color_tl ,
1022   color .value_required:n = true ,
1023 }

```

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

```

1024 \keys_define:nn { nicematrix }
1025 {
1026   NiceMatrixOptions .inherit:n =
1027     { nicematrix / Global } ,
1028   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1029   NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1030   NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1031   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1032   SubMatrix / rules .inherit:n = nicematrix / rules ,
1033   CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1034   CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1035   CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1036   NiceMatrix .inherit:n =
1037     {
1038       nicematrix / Global ,
1039       nicematrix / environments ,
1040     } ,
1041   NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1042   NiceMatrix / rules .inherit:n = nicematrix / rules ,
1043   NiceTabular .inherit:n =
1044     {
1045       nicematrix / Global ,
1046       nicematrix / environments
1047     } ,
1048   NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1049   NiceTabular / rules .inherit:n = nicematrix / rules ,
1050   NiceTabular / notes .inherit:n = nicematrix / notes ,
1051   NiceArray .inherit:n =
1052     {
1053       nicematrix / Global ,
1054       nicematrix / environments ,
1055     } ,
1056   NiceArray / xdots .inherit:n = nicematrix / xdots ,
1057   NiceArray / rules .inherit:n = nicematrix / rules ,
1058   pNiceArray .inherit:n =
1059     {
1060       nicematrix / Global ,
1061       nicematrix / environments ,
1062     } ,
1063   pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1064   pNiceArray / rules .inherit:n = nicematrix / rules ,
1065 }

```

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

```

1066 \keys_define:nn { nicematrix / NiceMatrixOptions }
1067 {
1068   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1069   delimiters / color .value_required:n = true ,
1070   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1071   delimiters / max-width .default:n = true ,
1072   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1073   delimiters .value_required:n = true ,

```

```

1074 width .dim_set:N = \l_@@_width_dim ,
1075 width .value_required:n = true ,
1076 last-col .code:n =
1077   \tl_if_empty:nF { #1 }
1078     { \@@_error:n { last-col-non-empty-for~NiceMatrixOptions } }
1079   \int_zero:N \l_@@_last_col_int ,
1080 small .bool_set:N = \l_@@_small_bool ,
1081 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.

```

1082 renew-matrix .code:n = \@@_renew_matrix: ,
1083 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}`.

```

1084 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.

```

1085 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)`.

```

1086   \str_if_eq:eeTF { #1 } { auto }
1087   { \@@_error:n { Option~auto~for~columns-width } }
1088   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1089 allow-duplicate-names .code:n =
1090   \cs_set:Nn \@@_err_duplicate_names:n { } ,
1091 allow-duplicate-names .value_forbidden:n = true ,
1092 notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1093 notes .value_required:n = true ,
1094 sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1095 sub-matrix .value_required:n = true ,
1096 matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1097 matrix / columns-type .value_required:n = true ,
1098 caption-above .bool_set:N = \l_@@_caption_above_bool ,
1099 caption-above .default:n = true ,
1100 unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1101 }

```

`\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.

```

1102 \NewDocumentCommand \NiceMatrixOptions { m }
1103 { \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.

```

1104 \keys_define:nn { nicematrix / NiceMatrix }
1105 {
1106   last-col .code:n = \tl_if_empty:nTF { #1 }
1107     {
1108       \bool_set_true:N \l_@@_last_col_without_value_bool
1109       \int_set:Nn \l_@@_last_col_int { -1 }
1110     }
1111   { \int_set:Nn \l_@@_last_col_int { #1 } } ,

```

```

1112 columns-type .tl_set:N = \l_@@_columns_type_tl ,
1113 columns-type .value_required:n = true ,
1114 l .meta:n = { columns-type = l } ,
1115 r .meta:n = { columns-type = r } ,
1116 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1117 delimiters / color .value_required:n = true ,
1118 delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1119 delimiters / max-width .default:n = true ,
1120 delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1121 delimiters .value_required:n = true ,
1122 small .bool_set:N = \l_@@_small_bool ,
1123 small .value_forbidden:n = true ,
1124 unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1125 }

```

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

```

1126 \keys_define:nn { nicematrix / NiceArray }
1127 {

```

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.

```

1128 small .bool_set:N = \l_@@_small_bool ,
1129 small .value_forbidden:n = true ,
1130 last-col .code:n = \tl_if_empty:nF { #1 }
1131 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1132 \int_zero:N \l_@@_last_col_int ,
1133 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1134 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1135 unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1136 }
1137 \keys_define:nn { nicematrix / pNiceArray }
1138 {
1139 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1140 last-col .code:n = \tl_if_empty:nF { #1 }
1141 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1142 \int_zero:N \l_@@_last_col_int ,
1143 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1144 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1145 delimiters / color .value_required:n = true ,
1146 delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1147 delimiters / max-width .default:n = true ,
1148 delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1149 delimiters .value_required:n = true ,
1150 small .bool_set:N = \l_@@_small_bool ,
1151 small .value_forbidden:n = true ,
1152 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1153 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1154 unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1155 }

```

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

```

1156 \keys_define:nn { nicematrix / NiceTabular }
1157 {

```

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.

```

1158 width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1159 \bool_set_true:N \l_@@_width_used_bool ,
1160 width .value_required:n = true ,

```

```

1161 notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1162 tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1163 tabularnote .value_required:n = true ,
1164 caption .tl_set:N = \l_@@_caption_tl ,
1165 caption .value_required:n = true ,
1166 short-caption .tl_set:N = \l_@@_short_caption_tl ,
1167 short-caption .value_required:n = true ,
1168 label .tl_set:N = \l_@@_label_tl ,
1169 label .value_required:n = true ,
1170 last-col .code:n = \tl_if_empty:nF { #1 }
1171 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1172 \int_zero:N \l_@@_last_col_int ,
1173 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1174 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1175 unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
1176 }

```

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
```

```

1177 \keys_define:nn { nicematrix / CodeAfter }
1178 {
1179   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1180   delimiters / color .value_required:n = true ,
1181   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1182   rules .value_required:n = true ,
1183   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1184   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1185   sub-matrix .value_required:n = true ,
1186   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1187 }

```

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}`).

```

1188 \cs_new_protected:Npn \@@_cell_begin:
1189 {

```

`\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.

```
1190   \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).

```
1191   \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.

```
1192   \cs_set_eq:NN \Hline \@@_Hline_in_cell:
```

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

```
1193   \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.

```

1194   \int_compare:nNt { \c@jCol } = { \c_one_int }
1195   {
1196     \int_compare:nNt { \l_@@_first_col_int } = { \c_one_int }
1197     { \@@_begin_of_row: }
1198   }

```

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:`.

```

1199   \hbox_set:Nw \l_@@_cell_box

```

The following command is nullified in the tabulars.

```

1200   \@@_tuning_not_tabular_begin:
1201   \@@_tuning_first_row:
1202   \@@_tuning_last_row:
1203   \g_@@_row_style_tl
1204 }

```

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

```

\cs_new_protected:Npn \@@_tuning_first_row:
{
  \int_if_zero:nT { \c@iRow }
  {
    \int_if_zero:nF { \c@jCol }
    {
      \l_@@_code_for_first_row_tl
      \xglobal \colorlet { nicematrix-first-row } { . }
    }
  }
}

```

We will use a version a little more efficient.

```

1205 \cs_new_protected:Npn \@@_tuning_first_row:
1206 {
1207   \if_int_compare:w \c@iRow = \c_zero_int
1208   \if_int_compare:w \c@jCol > \c_zero_int
1209   \l_@@_code_for_first_row_tl
1210   \xglobal \colorlet { nicematrix-first-row } { . }
1211   \fi:
1212   \fi:
1213 }

```

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

```

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

```

1214 \cs_new_protected:Npn \@@_tuning_last_row:
1215 {
1216   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1217   \l_@@_code_for_last_row_tl

```

```

1218     \xglobal \colorlet { nicematrix-last-row } { . }
1219     \fi:
1220 }

```

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

```

1221 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1222 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1223 {
1224     \m@th
1225     \c_math_toggle_token

```

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

```

1226     \@@_tuning_key_small:
1227 }
1228 \cs_set_eq:NN \@@_tuning_not_tabular_end: \c_math_toggle_token

```

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

```

1229 \cs_new_protected:Npn \@@_begin_of_row:
1230 {
1231     \int_gincr:N \c@iRow
1232     \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1233     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1234     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1235     \pgfpicture
1236     \pgfrememberpicturepositiononpagetrue
1237     \pgfcoordinate
1238     { \@@_env: - row - \int_use:N \c@iRow - base }
1239     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1240     \str_if_empty:NF \l_@@_name_str
1241     {
1242         \pgfnodealias
1243         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1244         { \@@_env: - row - \int_use:N \c@iRow - base }
1245     }
1246     \endpgfpicture
1247 }

```

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.

```

1248 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1249 {
1250     \int_if_zero:nTF { \c@iRow }
1251     {
1252         \dim_compare:nNnT
1253         { \box_dp:N \l_@@_cell_box } > { \g_@@_dp_row_zero_dim }
1254         { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1255         \dim_compare:nNnT
1256         { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_zero_dim }
1257         { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1258     }
1259     {
1260         \int_compare:nNnT { \c@iRow } = { \c_one_int }
1261         {
1262             \dim_compare:nNnT
1263             { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_one_dim }
1264             { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }

```

```

1265     }
1266   }
1267 }

1268 \cs_new_protected:Npn \@@_rotate_cell_box:
1269 {
1270   \box_rotate:Nn \l_@@_cell_box { 90 }
1271   \bool_if:NTF \g_@@_rotate_c_bool
1272   {
1273     \hbox_set:Nn \l_@@_cell_box
1274     {
1275       \m@th
1276       \c_math_toggle_token
1277       \vcenter { \box_use:N \l_@@_cell_box }
1278       \c_math_toggle_token
1279     }
1280   }
1281   {
1282     \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
1283     {
1284       \vbox_set_top:Nn \l_@@_cell_box
1285       {
1286         \vbox_to_zero:n { }
1287         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1288         \box_use:N \l_@@_cell_box
1289       }
1290     }
1291   }
1292   \bool_gset_false:N \g_@@_rotate_bool
1293   \bool_gset_false:N \g_@@_rotate_c_bool
1294 }

1295 \cs_new_protected:Npn \@@_adjust_size_box:
1296 {
1297   \dim_compare:nNnT { \g_@@_blocks_wd_dim } > { \c_zero_dim }
1298   {
1299     \box_set_wd:Nn \l_@@_cell_box
1300     { \dim_max:nn { \box_wd:N \l_@@_cell_box } { \g_@@_blocks_wd_dim } }
1301     \dim_gzero:N \g_@@_blocks_wd_dim
1302   }
1303   \dim_compare:nNnT { \g_@@_blocks_dp_dim } > { \c_zero_dim }
1304   {
1305     \box_set_dp:Nn \l_@@_cell_box
1306     { \dim_max:nn { \box_dp:N \l_@@_cell_box } { \g_@@_blocks_dp_dim } }
1307     \dim_gzero:N \g_@@_blocks_dp_dim
1308   }
1309   \dim_compare:nNnT { \g_@@_blocks_ht_dim } > { \c_zero_dim }
1310   {
1311     \box_set_ht:Nn \l_@@_cell_box
1312     { \dim_max:nn { \box_ht:N \l_@@_cell_box } { \g_@@_blocks_ht_dim } }
1313     \dim_gzero:N \g_@@_blocks_ht_dim
1314   }
1315 }

1316 \cs_new_protected:Npn \@@_cell_end:
1317 {

```

The following command is nullified in the tabulars.

```

1318   \@@_tuning_not_tabular_end:
1319   \hbox_set_end:
1320   \@@_cell_end_i:
1321 }

1322 \cs_new_protected:Npn \@@_cell_end_i:
1323 {

```

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.

```

1324 \g_@@_cell_after_hook_tl
1325 \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
1326 \@@_adjust_size_box:
1327 \box_set_ht:Nn \l_@@_cell_box
1328 { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1329 \box_set_dp:Nn \l_@@_cell_box
1330 { \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”).

```

1331 \@@_update_max_cell_width:

```

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

```

1332 \@@_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.

```

1333 \bool_if:NTF \g_@@_empty_cell_bool
1334 { \box_use_drop:N \l_@@_cell_box }
1335 {
1336   \bool_if:NTF \g_@@_not_empty_cell_bool
1337   { \@@_print_node_cell: }
1338   {
1339     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
1340     { \@@_print_node_cell: }
1341     { \box_use_drop:N \l_@@_cell_box }
1342   }
1343 }
1344 \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
1345 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1346 \bool_gset_false:N \g_@@_empty_cell_bool
1347 \bool_gset_false:N \g_@@_not_empty_cell_bool
1348 }

```

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

```

1349 \cs_new_protected:Npn \@@_update_max_cell_width:
1350 {
1351   \dim_gset:Nn \g_@@_max_cell_width_dim
1352   { \dim_max:nn { \g_@@_max_cell_width_dim } { \box_wd:N \l_@@_cell_box } }
1353 }

```


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`).

```

1354 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1355 {
1356   \@@_math_toggle:
1357   \hbox_set_end:
1358   \bool_if:NF \g_@@_rotate_bool
1359   {
1360     \hbox_set:Nn \l_@@_cell_box
1361     {
1362       \makebox [ \l_@@_col_width_dim ] [ s ]
1363       { \hbox_unpack_drop:N \l_@@_cell_box }
1364     }
1365   }
1366   \@@_cell_end_i:
1367 }

1368 \pgfset
1369 {
1370   nicematrix / cell-node /.style =
1371   {
1372     inner-sep = \c_zero_dim ,
1373     minimum-width = \c_zero_dim
1374   }
1375 }
```

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.

```

1376 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1377 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1378 {
1379   \use:c
1380   {
1381     __siunitx_table_align_
1382     \bool_if:NTF \l__siunitx_table_text_bool
1383     { \l__siunitx_table_align_text_tl }
1384     { \l__siunitx_table_align_number_tl }
1385     :n
1386   }
1387   { #1 }
1388 }
```

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!

```

1389 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1390 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1391 {
1392   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1393   \hbox:n
1394   {
1395     \pgfsys@markposition
1396     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1397   }
1398   #1
1399   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1400   \hbox:n
1401   {
1402     \pgfsys@markposition
```

```

1403         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1404     }
1405 }

1406 \cs_new_protected:Npn \@@_print_node_cell:
1407 {
1408     \socket_use:nn { nicematrix / siunitx-wrap }
1409     { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1410 }

```

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

```

1411 \cs_new_protected:Npn \@@_node_cell:
1412 {
1413     \pgfpicture
1414     \pgfsetbaseline \c_zero_dim
1415     \pgfrememberpicturepositiononpagetrue
1416     \pgfset { nicematrix / cell-node }
1417     \pgfnode
1418     { rectangle }
1419     { base }
1420     {

```

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).

```

1421     \sys_if_engine_xetex:T { \set@color }
1422     \box_use:N \l_@@_cell_box
1423 }
1424 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1425 { \l_@@_pgf_node_code_tl }
1426 \str_if_empty:NF \l_@@_name_str
1427 {
1428     \pgfnodealias
1429     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1430     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1431 }
1432 \endpgfpicture
1433 }

```

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).

```

1434 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1435 {
1436     \bool_if:nTF { #1 } { \tl_gput_left:ce } { \tl_gput_right:ce }

```

```

1437     { g_@@_ #2 _ lines _ t1 }
1438     {
1439         \use:c { @@ _ draw _ #2 : nnn }
1440         { \int_use:N \c@iRow }
1441         { \int_use:N \c@jCol }
1442         { \exp_not:n { #3 } }
1443     }
1444 }

1445 \cs_new_protected:Npn \@@_array:n
1446 {
1447     \dim_set:Nn \col@sep
1448     { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1449     \dim_compare:nNnTF { \l_@@_tabular_width_dim } = { \c_zero_dim }
1450     { \def \@halignto { } }
1451     { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1452     \@tabarray

```

\l_@@_baseline_t1 may have the value t, c or b. However, if the value is b, we compose the \array (of array) with the option t and the right translation will be done further. Remark that \str_if_eq:eeTF is fully expandable and we need something fully expandable here. \str_if_eq:ee(TF) is faster than \str_if_eq:nn(TF).

```

1453     [ \str_if_eq:eeTF \l_@@_baseline_t1 { c } { c } { t } ]
1454 }
1455 \cs_generate_variant:Nn \@@_array:n { o }

```

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

```

1456 \bool_if:NTF \c_@@_revtex_bool
1457 { \cs_set_eq:NN \@@_old_ialign: \ialign }

```

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

```

1458 { \cs_set_eq:cN { @@_old_ar@ialign: } \ar@ialign }

```

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

```

1459 \cs_new_protected:Npn \@@_create_row_node:
1460 {
1461     \int_compare:nNnT { \c@iRow } > { \g_@@_last_row_node_int }
1462     {
1463         \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1464         \@@_create_row_node_i:
1465     }
1466 }

1467 \cs_new_protected:Npn \@@_create_row_node_i:
1468 {

```

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

```

1469     \hbox
1470     {
1471         \bool_if:NT \l_@@_code_before_bool
1472         {
1473             \vtop
1474             {
1475                 \skip_vertical:N 0.5\arrayrulewidth
1476                 \pgfsys@markposition
1477                 { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1478                 \skip_vertical:N -0.5\arrayrulewidth
1479             }
1480         }

```

```

1481 \pgfpicture
1482 \pgfrememberpicturepositiononpagetrue
1483 \pgfcoordinate { \l_@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1484 { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1485 \str_if_empty:NF \l_@@_name_str
1486 {
1487 \pgfnodealias
1488 { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1489 { \l_@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1490 }
1491 \endpgfpicture
1492 }
1493 }

```

```

1494 \cs_new_protected:Npn \l_@@_in_every_cr:
1495 {
1496 \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1497 \tbl_update_cell_data_for_next_row:
1498 \int_gzero:N \c@jCol
1499 \bool_gset_false:N \g_@@_after_col_zero_bool
1500 \bool_if:NF \g_@@_row_of_col_done_bool
1501 {
1502 \l_@@_create_row_node:

```

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

```

1503 \clist_if_empty:NF \l_@@_hlines_clist
1504 {
1505 \str_if_eq:eeF \l_@@_hlines_clist { all }
1506 {
1507 \clist_if_in:NeT
1508 \l_@@_hlines_clist
1509 { \int_eval:n { \c@iRow + 1 } }
1510 }
1511 {

```

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.

```

1512 \int_compare:nNnT { \c@iRow } > { -1 }
1513 {
1514 \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
1515 { \hrule height \arrayrulewidth width \c_zero_dim }
1516 }
1517 }
1518 }
1519 }
1520 }

```

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

```

1521 \cs_set_protected:Npn \l_@@_renew_dots:
1522 {
1523 \cs_set_eq:NN \ldots \l_@@_Ldots:
1524 \cs_set_eq:NN \cdots \l_@@_Cdots:
1525 \cs_set_eq:NN \vdots \l_@@_Vdots:
1526 \cs_set_eq:NN \ddots \l_@@_Ddots:
1527 \cs_set_eq:NN \iddots \l_@@_Iddots:
1528 \cs_set_eq:NN \dots \l_@@_Ldots:
1529 \cs_set_eq:NN \hdotsfor \l_@@_Hdotsfor:
1530 }

```

If `booktabs` is loaded, we have to patch the macro `\@BTnormal` which is a macro of `booktabs`. The macro `\@BTnormal` draws an horizontal rule but it occurs after a vertical skip done by a low level TeX command. When this macro `\@BTnormal` occurs, the `row` node has yet been inserted by `nicematrix` *before* the vertical skip (and thus, at a wrong place). That's why we decide to create a new `row` node (for the same row). We patch the macro `\@BTnormal` to create this `row` node. This new `row` node will overwrite the previous definition of that `row` node and we have managed to avoid the error messages of that redefinition⁵.

```

1531 \hook_gput_code:nnn { begindocument } { . }
1532 {
1533   \IfPackageLoadedTF { booktabs }
1534   {
1535     \cs_new_protected:Npn \@_patch_booktabs:
1536       { \tl_put_left:Nn \@BTnormal \@_create_row_node_i: }
1537   }
1538   { \cs_new_protected:Npn \@_patch_booktabs: { } }
1539 }

```

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`.

```

1540 \cs_new_protected:Npn \@_some_initialization:
1541 {
1542   \@_everycr:
1543   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1544   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1545   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1546   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1547   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1548   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1549 }

```

```

1550 \cs_new_protected:Npn \@_pre_array_ii:
1551 {

```

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

```

1552   \fp_gzero:N \g_@@_total_X_weight_fp
1553   \bool_gset_false:N \g_@@_V_of_X_bool
1554   \@_expand_clist:N \l_@@_hlines_clist
1555   \@_expand_clist:N \l_@@_vlines_clist
1556   \@_patch_booktabs:
1557   \box_clear_new:N \l_@@_cell_box
1558   \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}`).

```

1559   \bool_if:NT \l_@@_small_bool
1560   {
1561     \def \arraystretch { 0.47 }
1562     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

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

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

```
1563     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1564 }
```

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!

```
1565 \bool_if:NT \g_@@_create_cell_nodes_bool
1566 {
1567   \tl_put_right:Nn \@@_begin_of_row:
1568   {
1569     \pgfsys@markposition
1570     { \@@_env: - row - \int_use:N \c@iRow - base }
1571   }
1572   \socket_assign_plug:n { nicematrix / create-cell-nodes } { active }
1573 }
```

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

```
1574 \bool_if:NF \c_@@_revtex_bool
1575 {
1576   \def \ar@ialign
1577   {
1578     \IfPackageLoadedT { latex-lab-testphase-table }
1579     { \tbl_init_cell_data_for_table: }
1580     \@@_some_initialization:
1581     \dim_zero:N \tabskip
```

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

```
1582     \cs_set_eq:Nc \ar@ialign { @@_old_ar@ialign: }
1583     \halign
1584   }
1585 }
```

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).

```
1586 \bool_if:NT \c_@@_revtex_bool
1587 {
1588   \IfPackageLoadedT { colortbl }
1589   { \cs_set_protected:Npn \CT@setup { } }
1590 }
```

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).

```
1591 \cs_set_eq:NN \@@_old_ldots: \ldots
1592 \cs_set_eq:NN \@@_old_cdots: \cdots
1593 \cs_set_eq:NN \@@_old_vdots: \vdots
1594 \cs_set_eq:NN \@@_old_ddots: \ddots
1595 \cs_set_eq:NN \@@_old_iddots: \iddots
1596 \bool_if:NTF \l_@@_standard_cline_bool
1597 { \cs_set_eq:NN \cline \@@_standard_cline: }
1598 { \cs_set_eq:NN \cline \@@_cline: }
1599 \cs_set_eq:NN \Ldots \@@_Ldots:
1600 \cs_set_eq:NN \Cdots \@@_Cdots:
1601 \cs_set_eq:NN \Vdots \@@_Vdots:
```

```

1602 \cs_set_eq:NN \Ddots \@@_Ddots:
1603 \cs_set_eq:NN \Iddots \@@_Iddots:
1604 \cs_set_eq:NN \Hline \@@_Hline:
1605 \cs_set_eq:NN \Hspace \@@_Hspace:
1606 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1607 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1608 \cs_set_eq:NN \Block \@@_Block:
1609 \cs_set_eq:NN \rotate \@@_rotate:
1610 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1611 \cs_set_eq:NN \dotfill \@@_dotfill:
1612 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1613 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1614 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1615 \cs_set_eq:NN \TopRule \@@_TopRule
1616 \cs_set_eq:NN \MidRule \@@_MidRule
1617 \cs_set_eq:NN \BottomRule \@@_BottomRule
1618 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1619 \cs_set_eq:NN \Hbrace \@@_Hbrace
1620 \cs_set_eq:NN \Vbrace \@@_Vbrace
1621 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1622 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1623 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1624 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1625 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1626 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1627 \int_compare:nNt { \l_@@_first_row_int } > { \c_zero_int }
1628 { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1629 \int_compare:nNt { \l_@@_last_row_int } < { \c_zero_int }
1630 { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1631 \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:`.

```

1632 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1633 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1634 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1635 \@@_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!).

```

1636 \tl_if_exist:NT \l_@@_note_in_caption_tl
1637 {
1638   \tl_if_empty:NF \l_@@_note_in_caption_tl
1639   {
1640     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1641     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1642   }
1643 }

```

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).

```

1644 \seq_gclear:N \g_@@_multicolumn_cells_seq
1645 \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).

```

1646 \int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }

```

At the end of the environment `{array}`, `\c@iRow` will be the total number de rows.
`\g_@@_row_total_int` will be the number or rows excepted the last row (if `\l_@@_last_row_bool` has been raised with the option `last-row`).

```
1647 \int_gzero_new:N \g_@@_row_total_int
```

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

```
1648 \int_gzero_new:N \g_@@_col_total_int
1649 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1650 \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.

```
1651 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1652 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1653 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1654 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1655 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1656 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1657 \tl_gclear:N \g_nicematrix_code_before_tl
1658 \tl_gclear:N \g_@@_pre_code_before_tl
1659 }
```

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

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

```
1660 \cs_new_protected:Npn \@@_pre_array:
1661 {
1662   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1663   \int_gzero_new:N \c@iRow
1664   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1665   \int_gzero_new:N \c@jCol
```

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

```
1666 \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
1667 {
1668   \bool_set_true:N \l_@@_last_row_without_value_bool
1669   \bool_if:NT \g_@@_aux_found_bool
1670     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1671 }
1672 \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
1673 {
1674   \bool_if:NT \g_@@_aux_found_bool
1675     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1676 }
```

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”.

```
1677 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
1678 {
1679   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1680     {
1681       \dim_compare:nNnT { \g_@@_ht_last_row_dim } < { \box_ht:N \l_@@_cell_box }
```



```

1682         { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1683         \dim_compare:nNnT { \g_@@_dp_last_row_dim } < { \box_dp:N \l_@@_cell_box }
1684         { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1685     }
1686 }

1687 \seq_gclear:N \g_@@_cols_vlism_seq
1688 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1689 \bool_if:NT \l_@@_code_before_bool { \@@_exec_code_before: }

```

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

```

1690 \seq_gset_eq:NN \g_@@_pos_of_blocks_seq \g_@@_future_pos_of_blocks_seq
1691 \seq_gclear:N \g_@@_future_pos_of_blocks_seq

```

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

```

1692 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1693 \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.

```

1694 \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1695 \@@_pre_array_ii:

```

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

```

1696 \box_clear_new:N \l_@@_the_array_box

```

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

```

1697 \dim_zero_new:N \l_@@_left_delim_dim
1698 \dim_zero_new:N \l_@@_right_delim_dim
1699 \bool_if:NNTF \g_@@_delims_bool
1700 {

```

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

```

1701     \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1702     \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1703     \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1704     \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1705 }
1706 {
1707     \dim_gset:Nn \l_@@_left_delim_dim
1708     { 2 \bool_if:NNTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1709     \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1710 }

```

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

```

1711 \hbox_set:Nw \l_@@_the_array_box

```

```

1712 \skip_horizontal:N \l_@@_left_margin_dim
1713 \skip_horizontal:N \l_@@_extra_left_margin_dim
1714 \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).

```

1715 \m@th
1716 \c_math_toggle_token
1717 \bool_if:NTF \l_@@_light_syntax_bool
1718 { \use:c { @@-light-syntax } }
1719 { \use:c { @@-normal-syntax } }
1720 }

```

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

```

1721 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1722 {
1723   \tl_set:Nn \l_tmpa_tl { #1 }
1724   \int_compare:nNt { \char_value_catcode:n { 60 } } = { 13 }
1725   { \@@_rescan_for_spanish:N \l_tmpa_tl }
1726   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1727   \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...

```

1728 \@@_pre_array:
1729 }

```

9 The `\CodeBefore`

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

```

1730 \cs_new_protected:Npn \@@_pre_code_before:
1731 {

```

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

```

1732 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1733 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1734 \int_set:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1735 \int_set:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }

```

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

```

1736 \pgfsys@markposition { \@@_env: - position }
1737 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1738 \pgfpicture
1739 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1740 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int + 1 }
1741 {
1742   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1743   \pgfcoordinate { \@@_env: - row - ##1 }
1744   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1745 }

```

Now, the recreation of the col nodes.

```

1746 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int + 1 }
1747 {
1748   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1749   \pgfcoordinate { \@@_env: - col - ##1 }
1750   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1751 }

```

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

```

1752 \@@_create_diag_nodes:

```

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

```

1753 \bool_if:NT \g_@@_create_cell_nodes_bool { \@@_recreate_cell_nodes: }
1754 \endpgfpicture

```

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

```

1755 \@@_create_blocks_nodes:
1756 \IfPackageLoadedT { tikz }
1757 {
1758   \tikzset
1759   {
1760     every-picture / .style =
1761     { overlay , name~prefix = \@@_env: - }
1762   }
1763 }
1764 \cs_set_eq:NN \cellcolor \@@_cellcolor
1765 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1766 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1767 \cs_set_eq:NN \rowcolor \@@_rowcolor
1768 \cs_set_eq:NN \rowcolors \@@_rowcolors
1769 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1770 \cs_set_eq:NN \arraycolor \@@_arraycolor
1771 \cs_set_eq:NN \columncolor \@@_columncolor
1772 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1773 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1774 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1775 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1776 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1777 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1778 }

```

```

1779 \cs_new_protected:Npn \@@_exec_code_before:
1780 {

```

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...

```

1781 \clist_map_inline:Nn \l_@@_corners_cells_clist
1782 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1783 \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`.

```

1784 \@@_add_to_colors_seq:nn { { nocolor } } { }
1785 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1786 \group_begin:

```

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

```

1787 \if_mode_math:
1788   \@@_exec_code_before_i:
1789 \else:
1790   \c_math_toggle_token
1791   \@@_exec_code_before_i:
1792   \c_math_toggle_token
1793 \fi:
1794 \group_end:
1795 }

```

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`).

```

1796 \cs_new_protected:Npn \@@_exec_code_before_i:
1797 {
1798   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1799   { \@@_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`.

```

1800 \exp_last_unbraced:No \@@_CodeBefore_keys:
1801 \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.

```

1802 \@@_actually_color:
1803 \l_@@_code_before_tl
1804 \q_stop
1805 }

```

```

1806 \keys_define:nn { nicematrix / CodeBefore }
1807 {
1808   create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1809   create-cell-nodes .default:n = true ,
1810   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1811   sub-matrix .value_required:n = true ,
1812   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1813   delimiters / color .value_required:n = true ,
1814   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1815 }
1816 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1817 {
1818   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1819   \@@_CodeBefore:w
1820 }

```

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.

```

1821 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1822 {
1823   \bool_if:NT \g_@@_aux_found_bool
1824   {
1825     \@@_pre_code_before:
1826     \legacy_if:nF { measuring@ } { #1 }
1827   }
1828 }

```

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.

```

1829 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1830 {
1831   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
1832   {
1833     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1834     \pgfcoordinate { \@@_env: - row - ##1 - base }
1835     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1836     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
1837     {
1838       \cs_if_exist:cT
1839       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1840       {
1841         \pgfsys@getposition
1842         { \@@_env: - ##1 - #####1 - NW }
1843         \@@_node_position:
1844         \pgfsys@getposition
1845         { \@@_env: - ##1 - #####1 - SE }
1846         \@@_node_position_i:
1847         \@@_pgf_rect_node:nnn
1848         { \@@_env: - ##1 - #####1 }
1849         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1850         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1851       }
1852     }
1853   }
1854   \@@_create_extra_nodes:
1855   \@@_create_aliases_last:
1856 }

1857 \cs_new_protected:Npn \@@_create_aliases_last:
1858 {
1859   \int_step_inline:nn { \c@iRow }
1860   {
1861     \pgfnodealias
1862     { \@@_env: - ##1 - last }
1863     { \@@_env: - ##1 - \int_use:N \c@jCol }
1864   }
1865   \int_step_inline:nn { \c@jCol }
1866   {
1867     \pgfnodealias
1868     { \@@_env: - last - ##1 }
1869     { \@@_env: - \int_use:N \c@iRow - ##1 }
1870   }
1871   \pgfnodealias % added 2025-04-05
1872   { \@@_env: - last - last }
1873   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1874 }

1875 \cs_new_protected:Npn \@@_create_blocks_nodes:
1876 {
1877   \pgfpicture
1878   \pgf@relevantforpicturesizefalse
1879   \pgfrememberpicturepositiononpagetrue
1880   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1881   { \@@_create_one_block_node:nnnnn ##1 }
1882   \endpgfpicture
1883 }

```

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

```

1884 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1885 {
1886   \tl_if_empty:nF { #5 }
1887   {
1888     \@@_qpoint:n { col - #2 }
1889     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1890     \@@_qpoint:n { #1 }
1891     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1892     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1893     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1894     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1895     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1896     \@@_pgf_rect_node:nnnnn
1897     { \@@_env: - #5 }
1898     { \dim_use:N \l_tmpa_dim }
1899     { \dim_use:N \l_tmpb_dim }
1900     { \dim_use:N \l_@@_tmpc_dim }
1901     { \dim_use:N \l_@@_tmpd_dim }
1902   }
1903 }

1904 \cs_new_protected:Npn \@@_patch_for_revtext:
1905 {
1906   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1907   \cs_set_eq:NN \@array \@array@array
1908   \cs_set_eq:NN \@tabular \@tabular@array
1909   \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1910   \cs_set_eq:NN \array \array@array
1911   \cs_set_eq:NN \endarray \endarray@array
1912   \cs_set:Npn \endtabular { \endarray $\egroup } % $
1913   \cs_set_eq:NN \@mkpream \@mkpream@array
1914   \cs_set_eq:NN \@classx \@classx@array
1915   \cs_set_eq:NN \insert@column \insert@column@array
1916   \cs_set_eq:NN \@arraycr \@arraycr@array
1917   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1918   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1919 }

```

10 The environment `{NiceArrayWithDelims}`

```

1920 \NewDocumentEnvironment { NiceArrayWithDelims }
1921 { m m O { } m ! O { } t \CodeBefore }
1922 {
1923   \bool_if:NT \c_@@_revtex_bool { \@@_patch_for_revtext: }
1924   \@@_provide_pgfsyspdfmark:
1925   \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.

```

1926   \bgroup

1927   \tl_gset:Nn \g_@@_left_delim_tl { #1 }

```

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).

```

1928 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1929 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1930 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

1931 \int_gzero:N \g_@@_block_box_int
1932 \dim_gzero:N \g_@@_width_last_col_dim
1933 \dim_gzero:N \g_@@_width_first_col_dim
1934 \bool_gset_false:N \g_@@_row_of_col_done_bool
1935 \str_if_empty:NT \g_@@_name_env_str
1936 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1937 \bool_if:NTF \l_@@_tabular_bool
1938 { \mode_leave_vertical: }
1939 { \@@_test_if_math_mode: }
1940 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1941 \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.

```

1942 \cs_gset_eq:cN { @@_old_CT@arc@ } \CT@arc@

```

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

```

1943 \cs_if_exist:NT \tikz@library@external@loaded
1944 {
1945   \tikzexternaldisable
1946   \cs_if_exist:NT \ifstandalone
1947     { \tikzset { external / optimize = false } }
1948 }

```

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

```

1949 \int_gincr:N \g_@@_env_int
1950 \bool_if:NF \l_@@_block_auto_columns_width_bool
1951 { \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.

```

1952 \seq_gclear:N \g_@@_blocks_seq
1953 \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`.

```

1954 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1955 \seq_gclear:N \g_@@_pos_of_xdots_seq
1956 \tl_gclear_new:N \g_@@_code_before_tl
1957 \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.

```

1958 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1959 {
1960   \bool_gset_true:N \g_@@_aux_found_bool
1961   \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1962 }
1963 { \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.

```

1964 \tl_gclear:N \g_@@_aux_tl

```

⁸e.g. `\color[rgb]{0.5,0.5,0}`

```

1965 \tl_if_empty:NF \g_@@_code_before_tl
1966 {
1967     \bool_set_true:N \l_@@_code_before_bool
1968     \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1969 }
1970 \tl_if_empty:NF \g_@@_pre_code_before_tl
1971 { \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`.

```

1972 \bool_if:NTF \g_@@_delims_bool
1973 { \keys_set:nn { nicematrix / pNiceArray } }
1974 { \keys_set:nn { nicematrix / NiceArray } }
1975 { #3 , #5 }

1976 \@@_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:.`

```

1977 \bool_if:nTF { #6 } { \@@_CodeBefore_Body:w } { \@@_pre_array: }
1978 }

```

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

```

1979 {
1980     \bool_if:NTF \l_@@_light_syntax_bool
1981     { \use:c { end @@-light-syntax } }
1982     { \use:c { end @@-normal-syntax } }
1983     \c_math_toggle_token
1984     \skip_horizontal:N \l_@@_right_margin_dim
1985     \skip_horizontal:N \l_@@_extra_right_margin_dim
1986     \hbox_set_end:
1987     \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.

```

1988 \bool_if:NT \l_@@_width_used_bool
1989 {
1990     \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
1991     { \@@_error_or_warning:n { width-without-X-columns } }
1992 }

```

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 .

```

1993 \fp_compare:nNnT { \g_@@_total_X_weight_fp } > { \c_zero_fp }
1994 { \@@_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).

```

1995 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
1996 {
1997     \bool_if:NF \l_@@_last_row_without_value_bool
1998     {
1999         \int_compare:nNnF { \l_@@_last_row_int } = { \c@iRow }
2000         {
2001             \@@_error:n { Wrong-last-row }
2002             \int_gset_eq:NN \l_@@_last_row_int \c@iRow

```



```

2003         }
2004     }
2005 }

```

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

```

2006     \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2007     \bool_if:NTF \g_@@_last_col_found_bool
2008     { \int_gdecr:N \c@jCol }
2009     {
2010         \int_compare:nNnT { \l_@@_last_col_int } > { -1 }
2011         { \@@_error:n { last~col~not~used } }
2012     }

```

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

```

2013     \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2014     \int_compare:nNnT { \l_@@_last_row_int } > { -1 }
2015     { \int_gdecr:N \c@iRow }

```

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

```

2016     \int_if_zero:nT { \l_@@_first_col_int }
2017     { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2018     \bool_if:nTF { ! \g_@@_delims_bool }
2019     {
2020         \str_if_eq:eeTF \l_@@_baseline_tl { c }
2021         { \@@_use_arraybox_with_notes_c: }
2022         {
2023             \str_if_eq:eeTF \l_@@_baseline_tl { b }
2024             { \@@_use_arraybox_with_notes_b: }
2025             { \@@_use_arraybox_with_notes: }
2026         }
2027     }

```

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).

```

2028     {
2029         \int_if_zero:nTF { \l_@@_first_row_int }
2030         {
2031             \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2032             \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2033         }
2034         { \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

```

2035         \int_compare:nNnTF { \l_@@_last_row_int } > { -2 }
2036         {
2037             \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2038             \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2039         }
2040         { \dim_zero:N \l_tmpb_dim }
2041     \hbox_set:Nn \l_tmpa_box
2042     {
2043         \m@th
2044         \c_math_toggle_token
2045         \@@_color:o \l_@@_delimiters_color_tl

```

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

¹⁰A value of `-1` for `\l_@@_last_row_int` means that there is a “last row” but the the user have not set the value with the option `last row` (and we are in the first compilation).

```

2046      \exp_after:wN \left \g_@@_left_delim_tl
2047      \vcenter
2048      {

```

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.

```

2049      \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2050      \hbox
2051      {
2052          \bool_if:NTF \l_@@_tabular_bool
2053          { \skip_horizontal:n { - \tabcolsep } }
2054          { \skip_horizontal:n { - \arraycolsep } }
2055          \@@_use_arraybox_with_notes_c:
2056          \bool_if:NTF \l_@@_tabular_bool
2057          { \skip_horizontal:n { - \tabcolsep } }
2058          { \skip_horizontal:n { - \arraycolsep } }
2059      }

```

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

```

2060      \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2061      }
2062      \exp_after:wN \right \g_@@_right_delim_tl
2063      \c_math_toggle_token
2064      }

```

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.

```

2065      \bool_if:NTF \l_@@_delimiters_max_width_bool
2066      {
2067          \@@_put_box_in_flow_bis:nn
2068          { \g_@@_left_delim_tl }
2069          { \g_@@_right_delim_tl }
2070      }
2071      \@@_put_box_in_flow:
2072      }

```

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. 92).

```

2073      \bool_if:NT \g_@@_last_col_found_bool
2074      { \skip_horizontal:N \g_@@_width_last_col_dim }
2075      \bool_if:NT \l_@@_preamble_bool
2076      {
2077          \int_compare:nNnT { \c_jCol } < { \g_@@_static_num_of_col_int }
2078          { \@@_err_columns_not_used: }
2079      }
2080      \@@_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.

```

2081      \egroup

```

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

```

2082      \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2083      \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2084      \iow_now:Ne \@mainaux
2085      {
2086          \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2087          \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2088          { \exp_not:o \g_@@_aux_tl }
2089      }
2090      \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2091      \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2092      }

```

This is the end of the environment {NiceArrayWithDelims}.

```

2093 \cs_new_protected:Npn \@@_err_columns_not_used:
2094 {
2095   \@@_warning:n { columns~not~used }
2096   \cs_gset:Npn \@@_err_columns_not_used: { }
2097 }

```

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 .

```

2098 \cs_new_protected:Npn \@@_compute_width_X:
2099 {
2100   \tl_gput_right:Ne \g_@@_aux_tl
2101   {
2102     \bool_set_true:N \l_@@_X_columns_aux_bool
2103     \dim_set:Nn \l_@@_X_columns_dim
2104     {

```

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`

```

2105     \bool_lazy_and:nnTF
2106       { \g_@@_V_of_X_bool }
2107       { \l_@@_X_columns_aux_bool }
2108       { \dim_use:N \l_@@_X_columns_dim }
2109     {
2110       \dim_compare:nNnTF
2111         {
2112           \dim_abs:n
2113             { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2114         }
2115         <
2116         { 0.001 pt }
2117         { \dim_use:N \l_@@_X_columns_dim }
2118         {
2119           \dim_eval:n
2120             {
2121               \l_@@_X_columns_dim
2122               +
2123               \fp_to_dim:n
2124                 {
2125                   (
2126                     \dim_eval:n
2127                       { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2128                   )
2129                   / \fp_use:N \g_@@_total_X_weight_fp
2130                 }
2131             }
2132         }
2133       }
2134     }
2135   }
2136 }
2137 }

```

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`.

```

2138 \cs_new_protected:Npn \@@_transform_preamble:
2139 {
2140   \@@_transform_preamble_i:
2141   \@@_transform_preamble_ii:
2142 }
2143 \cs_new_protected:Npn \@@_transform_preamble_i:
2144 {
2145   \int_gzero:N \c@jCol

```

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

```

2146   \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.

```

2147   \bool_gset_false:N \g_tmpb_bool

```

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

```

2148   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2149   \int_zero:N \l_tmpa_int
2150   \tl_gclear:N \g_@@_array_preamble_tl
2151   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2152   {
2153     \tl_gset:Nn \g_@@_array_preamble_tl
2154     { ! { \skip_horizontal:N \arrayrulewidth } }
2155   }
2156   {
2157     \clist_if_in:NnT \l_@@_vlines_clist 1
2158     {
2159       \tl_gset:Nn \g_@@_array_preamble_tl
2160       { ! { \skip_horizontal:N \arrayrulewidth } }
2161     }
2162   }

```

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

```

2163   \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2164   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2165   \@@_replace_columncolor:
2166 }

2167 \cs_new_protected:Npn \@@_transform_preamble_ii:
2168 {

```

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

```

2169   \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2170   {
2171     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2172     { \bool_gset_true:N \g_@@_delims_bool }
2173   }
2174   { \bool_gset_true:N \g_@@_delims_bool }

```

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

```
2175 \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).

```
2176 \int_if_zero:nTF { \l_@@_first_col_int }
2177 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2178 {
2179   \bool_if:NF \g_@@_delims_bool
2180   {
2181     \bool_if:NF \l_@@_tabular_bool
2182     {
2183       \clist_if_empty:NT \l_@@_vlines_clist
2184       {
2185         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2186         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2187       }
2188     }
2189   }
2190 }
2191 \int_compare:nNnTF { \l_@@_last_col_int } > { -1 }
2192 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2193 {
2194   \bool_if:NF \g_@@_delims_bool
2195   {
2196     \bool_if:NF \l_@@_tabular_bool
2197     {
2198       \clist_if_empty:NT \l_@@_vlines_clist
2199       {
2200         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2201         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2202       }
2203     }
2204   }
2205 }
```

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

```
2206 \dim_compare:nNnT { \l_@@_tabular_width_dim } = { \c_zero_dim }
2207 {
```

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

```
2208 \IfPackageLoadedF { latex-lab-testphase-table }
2209 {
2210   \tl_gput_right:Nn \g_@@_array_preamble_tl
2211   { > { \@@_error_too_much_cols: } 1 }
2212 }
2213 }
2214 }
```

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.

```
2215 \cs_new_protected:Npn \@@_rec_preamble:n #1
2216 {
```

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

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.

```

2217 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2218 { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2219 {

```

Now, the columns defined by \newcolumnntype of array.

```

2220 \cs_if_exist:cTF { NC @ find @ #1 }
2221 {
2222 \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2223 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2224 }
2225 {
2226 \str_if_eq:nnTF { #1 } { S }
2227 { \@@_fatal:n { unknown~column~type~S } }
2228 { \@@_fatal:nn { unknown~column~type } { #1 } }
2229 }
2230 }
2231 }

```

For c, l and r

```

2232 \cs_new_protected:Npn \@@_c: #1
2233 {
2234 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2235 \tl_gclear:N \g_@@_pre_cell_tl
2236 \tl_gput_right:Nn \g_@@_array_preamble_tl
2237 { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2238 \int_gincr:N \c@jCol
2239 \@@_rec_preamble_after_col:n
2240 }
2241 \cs_new_protected:Npn \@@_l: #1
2242 {
2243 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2244 \tl_gclear:N \g_@@_pre_cell_tl
2245 \tl_gput_right:Nn \g_@@_array_preamble_tl
2246 {
2247 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2248 l
2249 < \@@_cell_end:
2250 }
2251 \int_gincr:N \c@jCol
2252 \@@_rec_preamble_after_col:n
2253 }
2254 \cs_new_protected:Npn \@@_r: #1
2255 {
2256 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2257 \tl_gclear:N \g_@@_pre_cell_tl
2258 \tl_gput_right:Nn \g_@@_array_preamble_tl
2259 {
2260 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2261 r
2262 < \@@_cell_end:
2263 }
2264 \int_gincr:N \c@jCol
2265 \@@_rec_preamble_after_col:n
2266 }

```

For ! and @

```

2267 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2268 {
2269 \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2270 \@@_rec_preamble:n
2271 }
2272 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2273 \cs_new_protected:cpn { @@ _ | : } #1
2274 {
\l_tmpa_int is the number of successive occurrences of |
2275     \int_incr:N \l_tmpa_int
2276     \@@_make_preamble_i_i:n
2277 }
2278 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2279 {

```

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

```

2280     \str_if_eq:nnTF { #1 } { | }
2281     { \use:c { @@ _ | : } | }
2282     { \@@_make_preamble_i_ii:nn { } #1 }
2283 }
2284 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2285 {
2286     \str_if_eq:nnTF { #2 } { [ ] }
2287     { \@@_make_preamble_i_ii:nw { #1 } [ ] }
2288     { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2289 }
2290 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2291 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2292 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2293 {
2294     \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2295     \tl_gput_right:Ne \g_@@_array_preamble_tl
2296     {

```

Here, the command \dim_use:N is mandatory.

```

2297     \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2298 }
2299 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2300 {
2301     \@@_vline:n
2302     {
2303         position = \int_eval:n { \c@jCol + 1 } ,
2304         multiplicity = \int_use:N \l_tmpa_int ,
2305         total-width = \dim_use:N \l_@@_rule_width_dim ,
2306         #2
2307     }

```

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.

```

2308     }
2309     \int_zero:N \l_tmpa_int
2310     \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2311     \@@_rec_preamble:n #1
2312 }

2313 \cs_new_protected:cpn { @@ _ > : } #1 #2
2314 {
2315     \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2316     \@@_rec_preamble:n
2317 }
2318 \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.

```

2319 \keys_define:nn { nicematrix / p-column }
2320 {
2321   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2322   r .value_forbidden:n = true ,
2323   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2324   c .value_forbidden:n = true ,
2325   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2326   l .value_forbidden:n = true ,
2327   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2328   S .value_forbidden:n = true ,
2329   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2330   p .value_forbidden:n = true ,
2331   t .meta:n = p ,
2332   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2333   m .value_forbidden:n = true ,
2334   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2335   b .value_forbidden:n = true
2336 }

```

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

```

2337 \cs_new_protected:Npn \@@_p: #1
2338 {
2339   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2340   \@@_make_preamble_ii_i:n
2341 }
2342 \cs_set_eq:NN \@@_b: \@@_p:
2343 \cs_set_eq:NN \@@_m: \@@_p:
2344 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2345 {
2346   \str_if_eq:nnTF { #1 } { [ ]
2347     { \@@_make_preamble_ii_ii:w [ ]
2348       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2349     }
2350 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2351 { \@@_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.

```

2352 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2353 {

```

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).

```

2354   \str_set:Nn \l_@@_hpos_col_str { j }
2355   \@@_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`.

```

2356   \setlength { \l_tmpa_dim } { #2 }
2357   \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2358 }
2359 \cs_new_protected:Npn \@@_keys_p_column:n #1
2360 { \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.

```
2361 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2362 {
```

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

```
2363 \use:e
2364 {
2365 \@@_make_preamble_ii_vi:nnnnnnnn
2366 { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }
2367 { #1 }
2368 {
```

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.

```
2369 \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2370 { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2371 {
```

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

```
2372 \def \exp_not:N \l_@@_hpos_cell_tl
2373 { \str_lowercase:f { \l_@@_hpos_col_str } }
2374 }
2375 \IfPackageLoadedTF { ragged2e }
2376 {
2377 \str_case:on \l_@@_hpos_col_str
2378 {
```

The following `\exp_not:N` are mandatory.

```
2379 c { \exp_not:N \Centering }
2380 l { \exp_not:N \RaggedRight }
2381 r { \exp_not:N \RaggedLeft }
2382 }
2383 }
2384 {
2385 \str_case:on \l_@@_hpos_col_str
2386 {
2387 c { \exp_not:N \centering }
2388 l { \exp_not:N \raggedright }
2389 r { \exp_not:N \raggedleft }
2390 }
2391 }
2392 #3
2393 }
2394 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2395 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2396 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2397 { #2 }
2398 {
2399 \str_case:onF \l_@@_hpos_col_str
2400 {
2401 { j } { c }
2402 { si } { c }
2403 }
```

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

```
2404 { \str_lowercase:f \l_@@_hpos_col_str }
2405 }
2406 }
```

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

```
2407 \int_gincr:N \c@jCol
2408 \@@_rec_preamble_after_col:n
2409 }
```

#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*.

```

2410 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2411 {
2412   \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2413   {
2414     \tl_gput_right:Nn \g_@@_array_preamble_tl
2415     { > \@@_test_if_empty_for_S: }
2416   }
2417   { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2418   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2419   \tl_gclear:N \g_@@_pre_cell_tl
2420   \tl_gput_right:Nn \g_@@_array_preamble_tl
2421   {
2422     > {

```

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.

```

2423   \dim_set:Nn \l_@@_col_width_dim { #2 }
2424   \IfPackageLoadedT { latex-lab-testphase-table }
2425   { \tag_struct_begin:n { tag = Div } }
2426   \@@_cell_begin:

```

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

```

2427   \use:c { #7 } [ #1 ] { #2 }

```

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

```

2428   \everypar
2429   {
2430     \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2431     \everypar { }
2432   }
2433   \IfPackageLoadedT { latex-lab-testphase-table }
2434   { \tagpdfpara0n }

```

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

```

2435   #3

```

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

```

2436   \g_@@_row_style_tl
2437   \arraybackslash
2438   #5
2439   }
2440   #8
2441   < {
2442   #6

```

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

```

2443   \@finalstrut \@arstrutbox
2444   \use:c { end #7 }

```

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

```

2445         #4
2446         \@@_cell_end:
2447         \IfPackageLoadedT { latex-lab-testphase-table }
2448             { \tag_struct_end: }
2449     }
2450 }
2451 }

```

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

```

2452 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2453 {

```

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 &).

```

2454     \group_align_safe_begin:
2455     \peek_meaning:NTF &
2456     { \@@_the_cell_is_empty: }
2457     {
2458         \peek_meaning:NTF \\\
2459         { \@@_the_cell_is_empty: }
2460         {
2461             \peek_meaning:NTF \crcr
2462             \@@_the_cell_is_empty:
2463             \group_align_safe_end:
2464         }
2465     }
2466 }
2467 \cs_new_protected:Npn \@@_the_cell_is_empty:
2468 {
2469     \group_align_safe_end:
2470     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2471     {

```

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

```

2472         \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2473         \skip_horizontal:N \l_@@_col_width_dim
2474     }
2475 }
2476 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2477 {
2478     \peek_meaning:NT \__siunitx_table_skip:n
2479     { \bool_gset_true:N \g_@@_empty_cell_bool }
2480 }

```

The following command will be used in m-columns in order to center vertically the box. In fact, despite its name, the command does not always center the cell. Indeed, if there is only one row in the cell, it should not be centered vertically. It's not possible to know the number of rows of the cell. However, we consider (as in array) that if the height of the cell is no more that the height of \strutbox, there is only one row.

```

2481 \cs_new_protected:Npn \@@_center_cell_box:
2482 {

```

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

```

2483     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2484     {
2485         \dim_compare:nNnT
2486             { \box_ht:N \l_@@_cell_box }
2487         >

```

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).

```

2488     { \box_ht:N \strutbox }
2489     {
2490       \hbox_set:Nn \l_@@_cell_box
2491       {
2492         \box_move_down:nn
2493         {
2494           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2495             + \baselineskip ) / 2
2496         }
2497         { \box_use:N \l_@@_cell_box }
2498       }
2499     }
2500   }
2501 }

```

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

```

2502 \cs_new_protected:Npn \@_V: #1 #2
2503 {
2504   \str_if_eq:nnTF { #2 } { [ ] }
2505   { \@@_make_preamble_V_i:w [ ] }
2506   { \@@_make_preamble_V_i:w [ ] { #2 } }
2507 }
2508 \cs_new_protected:Npn \@_make_preamble_V_i:w [ #1 ]
2509 { \@@_make_preamble_V_ii:nn { #1 } }
2510 \cs_new_protected:Npn \@_make_preamble_V_ii:nn #1 #2
2511 {
2512   \str_set:Nn \l_@@_vpos_col_str { p }
2513   \str_set:Nn \l_@@_hpos_col_str { j }
2514   \@@_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`.

```

2515   \setlength { \l_tmpa_dim } { #2 }
2516   \IfPackageLoadedTF { varwidth }
2517   { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2518   {
2519     \@@_error_or_warning:n { varwidth-not-loaded }
2520     \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2521   }
2522 }

```

For `w` and `W`

```

2523 \cs_new_protected:Npn \@_w: { \@@_make_preamble_w:nnnn { } }
2524 \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.

```

2525 \cs_new_protected:Npn \@_make_preamble_w:nnnn #1 #2 #3 #4
2526 {
2527   \str_if_eq:nnTF { #3 } { s }
2528   { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2529   { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2530 }

```

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.

```

2531 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2532 {
2533   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2534   \tl_gclear:N \g_@@_pre_cell_tl
2535   \tl_gput_right:Nn \g_@@_array_preamble_tl
2536   {
2537     > {

```

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`.

```

2538       \setlength { \l_@@_col_width_dim } { #2 }
2539       \@@_cell_begin:
2540       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2541     }
2542   c
2543   < {
2544     \@@_cell_end_for_w_s:
2545     #1
2546     \@@_adjust_size_box:
2547     \box_use_drop:N \l_@@_cell_box
2548   }
2549 }
2550 \int_gincr:N \c@jCol
2551 \@@_rec_preamble_after_col:n
2552 }

```

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

```

2553 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2554 {
2555   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2556   \tl_gclear:N \g_@@_pre_cell_tl
2557   \tl_gput_right:Nn \g_@@_array_preamble_tl
2558   {
2559     > {

```

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`.

```

2560       \setlength { \l_@@_col_width_dim } { #4 }
2561       \hbox_set:Nw \l_@@_cell_box
2562       \@@_cell_begin:
2563       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2564     }
2565   c
2566   < {
2567     \@@_cell_end:
2568     \hbox_set_end:
2569     #1
2570     \@@_adjust_size_box:
2571     \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2572   }
2573 }

```

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

```

2574   \int_gincr:N \c@jCol
2575   \@@_rec_preamble_after_col:n
2576 }

```

```

2577 \cs_new_protected:Npn \@@_special_W:
2578 {
2579   \dim_compare:nNt { \box_wd:N \l_@@_cell_box } > { \l_@@_col_width_dim }
2580   { \@@_warning:n { W~warning } }
2581 }

```

For S (of siunitx).

```

2582 \cs_new_protected:Npn \@@_S: #1 #2
2583 {
2584   \str_if_eq:nnTF { #2 } { [ ] }
2585   { \@@_make_preamble_S:w [ ] }
2586   { \@@_make_preamble_S:w [ ] { #2 } }
2587 }
2588 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2589 { \@@_make_preamble_S_i:n { #1 } }
2590 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2591 {
2592   \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2593   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2594   \tl_gclear:N \g_@@_pre_cell_tl
2595   \tl_gput_right:Nn \g_@@_array_preamble_tl
2596   {
2597     > {

```

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).

```

2598       \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2599       \keys_set:nn { siunitx } { #1 }
2600       \@@_cell_begin:
2601       \siunitx_cell_begin:w
2602     }
2603     c
2604     <
2605     {
2606       \siunitx_cell_end:

```

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

```

2607       \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2608       {
2609         \bool_if:NTF \l__siunitx_table_text_bool
2610         { \bool_set_true:N }
2611         { \bool_set_false:N }
2612         \l__siunitx_table_text_bool
2613       }
2614       \@@_cell_end:
2615     }
2616   }

```

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

```

2617       \int_gincr:N \c@jCol
2618       \@@_rec_preamble_after_col:n
2619     }

```

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

```

2620 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2621 {
2622   \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.

```

2623 \int_if_zero:nTF { \c@jCol }
2624 {
2625     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2626     {

```

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

```

2627         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2628         \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2629         \@@_rec_preamble:n #2
2630     }
2631     {
2632         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2633         \@@_make_preamble_iv:nn { #1 } { #2 }
2634     }
2635 }
2636 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2637 }
2638 \cs_set_eq:cc { @@ _ \token_to_str:N [ : ] { @@ _ \token_to_str:N ( : }
2639 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2640 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2641 {
2642     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2643     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2644     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2645     {
2646         \@@_error:nn { delimiter~after~opening } { #2 }
2647         \@@_rec_preamble:n
2648     }
2649     { \@@_rec_preamble:n #2 }
2650 }

```

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

```

2651 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2652 { \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}).

```

2653 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2654 {
2655     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2656     \tl_if_in:nnTF { ) ] \} } { #2 }
2657     { \@@_make_preamble_v:nnn #1 #2 }
2658     {
2659         \str_if_eq:nnTF { \s_stop } { #2 }
2660         {
2661             \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2662             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2663             {
2664                 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2665                 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2666                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2667                 \@@_rec_preamble:n #2
2668             }
2669         }
2670         {
2671             \tl_if_in:nnT { ( [ \{ \left } { #2 }
2672             { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2673             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2674             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }

```

```

2675         \@@_rec_preamble:n #2
2676     }
2677 }
2678 }
2679 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2680 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2681 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2682 {
2683     \str_if_eq:nnTF { \s_stop } { #3 }
2684     {
2685         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2686         {
2687             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2688             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2689             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2690             \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2691         }
2692         {
2693             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2694             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2695             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2696             \@@_error:nn { double~closing~delimiter } { #2 }
2697         }
2698     }
2699     {
2700         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2701         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2702         \@@_error:nn { double~closing~delimiter } { #2 }
2703         \@@_rec_preamble:n #3
2704     }
2705 }
2706 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2707 { \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 @{...}.

```

2708 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2709 {
2710     \str_if_eq:nnTF { #1 } { < }
2711     { \@@_rec_preamble_after_col_i:n }
2712     {
2713         \str_if_eq:nnTF { #1 } { @ }
2714         { \@@_rec_preamble_after_col_ii:n }
2715         {
2716             \str_if_eq:eeTF \l_@@_vlines_clist { all }
2717             {
2718                 \tl_gput_right:Nn \g_@@_array_preamble_tl
2719                 { ! { \skip_horizontal:N \arrayrulewidth } }
2720             }
2721             {
2722                 \clist_if_in:NeT \l_@@_vlines_clist
2723                 { \int_eval:n { \c@jCol + 1 } }
2724                 {
2725                     \tl_gput_right:Nn \g_@@_array_preamble_tl
2726                     { ! { \skip_horizontal:N \arrayrulewidth } }
2727                 }
2728             }
2729             \@@_rec_preamble:n { #1 }
2730         }
2731     }
2732 }

```



```

2733 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2734 {
2735   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2736   \@@_rec_preamble_after_col:n
2737 }

```

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.

```

2738 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2739 {
2740   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2741   {
2742     \tl_gput_right:Nn \g_@@_array_preamble_tl
2743     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2744   }
2745   {
2746     \clist_if_in:NneTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2747     {
2748       \tl_gput_right:Nn \g_@@_array_preamble_tl
2749       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2750     }
2751     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2752   }
2753   \@@_rec_preamble:n
2754 }

```

```

2755 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2756 {
2757   \tl_clear:N \l_tmpa_tl
2758   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2759   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2760 }

```

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

```

2761 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2762 { \@@_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.

```

2763 \cs_new_protected:Npn \@@_X: #1 #2
2764 {
2765   \str_if_eq:nnTF { #2 } { [ ]
2766     { \@@_make_preamble_X:w [ ] }
2767     { \@@_make_preamble_X:w [ ] #2 }
2768   }
2769   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2770   { \@@_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`.

```

2771 \keys_define:nn { nicematrix / X-column }
2772 {
2773   V .code:n =
2774   \IfPackageLoadedTF { varwidth }
2775   {
2776     \bool_set_true:N \l_@@_V_of_X_bool

```

```

2777         \bool_gset_true:N \g_@@_V_of_X_bool
2778     }
2779     { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
2780     unknown .code:n =
2781         \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2782         { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2783         { \@@_error_or_warning:n { invalid~weight } }
2784 }

```

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

```

2785 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2786 {

```

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).

```

2787     \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).

```

2788     \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`).

```

2789     \fp_set:Nn \l_tmpa_fp { 1.0 }
2790     \@@_keys_p_column:n { #1 }

```

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

```

2791     \bool_set_false:N \l_@@_V_of_X_bool
2792     \keys_set:no { nicematrix / X-column } \l_tmpa_tl

```

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

```

2793     \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).

```

2794     \bool_if:NTF \l_@@_X_columns_aux_bool
2795     {
2796         \@@_make_preamble_ii_iv:nnn

```

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

```

2797         { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2798         { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
2799         { \@@_no_update_width: }
2800     }

```

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.

```

2801     {
2802         \tl_gput_right:Nn \g_@@_array_preamble_tl
2803         {
2804             > {
2805                 \@@_cell_begin:
2806                 \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`.

```

2807         \NotEmpty

```

The following code will nullify the box of the cell.

```

2808         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2809         { \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`.

```

2810         \begin { minipage } { 5 cm } \arraybackslash
2811     }
2812     c
2813     < {
2814         \end { minipage }
2815         \@@_cell_end:
2816     }
2817 }
2818 \int_gincr:N \c@jCol
2819 \@@_rec_preamble_after_col:n
2820 }
2821 }

2822 \cs_new_protected:Npn \@@_no_update_width:
2823 {
2824     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2825     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2826 }

```

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

```

2827 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2828 {
2829     \seq_gput_right:Ne \g_@@_cols_vlism_seq
2830     { \int_eval:n { \c@jCol + 1 } }
2831     \tl_gput_right:Ne \g_@@_array_preamble_tl
2832     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2833     \@@_rec_preamble:n
2834 }

```

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.

```

2835 \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).

```

2836 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2837 { \@@_fatal:n { Preamble-forgotten } }
2838 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
2839 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
2840 { @@ _ \token_to_str:N \hline : }
2841 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
2842 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
2843 { @@ _ \token_to_str:N \hline : }
2844 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
2845 { @@ _ \token_to_str:N \hline : }
2846 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
2847 { @@ _ \token_to_str:N \hline : }
2848 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
2849 { @@ _ \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).

```

2850 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2851 {

```

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

```

2852 \multispan { #1 }
2853 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2854 \begingroup
2855 \IfPackageLoadedTF { latex-lab-testphase-table }
2856   { \tbl_update_multicolumn_cell_data:n { #1 } }
2857 \def \@@addamp
2858   { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2859 \tl_gclear:N \g_@@_preamble_tl
2860 \@@_make_m_preamble:n #2 \q_stop

```

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

```

2861 \exp_args:No \@mkpream \g_@@_preamble_tl
2862 \@@adddtopreamble \@empty
2863 \endgroup
2864 \UseTaggingSocket { tbl / colspan } { #1 }

```

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

```

2865 \int_compare:nNnT { #1 } > { \c_one_int }
2866 {
2867   \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2868     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2869   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2870   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2871     {
2872       {
2873         \int_if_zero:nTF { \c@jCol }
2874           { \int_eval:n { \c@iRow + 1 } }
2875           { \int_use:N \c@iRow }
2876       }
2877       { \int_eval:n { \c@jCol + 1 } }
2878       {
2879         \int_if_zero:nTF { \c@jCol }
2880           { \int_eval:n { \c@iRow + 1 } }
2881           { \int_use:N \c@iRow }
2882       }
2883       { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block

```

2884     { }
2885   }
2886 }

```

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

```

2887 \RenewDocumentCommand { \cellcolor } { 0 { } m }
2888 {
2889   \tl_gput_right:Ne \g_@@_pre_code_before_tl
2890     {
2891       \@@_rectanglecolor [ ##1 ]
2892       { \exp_not:n { ##2 } }
2893       { \int_use:N \c@iRow - \int_use:N \c@jCol }
2894       { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2895     }
2896   \ignorespaces
2897 }

```

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

```
2898 \def \@sharp { #3 }
2899 \@arstrut
2900 \@preamble
2901 \null
```

We add some lines.

```
2902 \int_gadd:Nn \c@jCol { #1 - 1 }
2903 \int_compare:nNt { \c@jCol } > { \g_@@_col_total_int }
2904 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2905 \ignorespaces
2906 }
```

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`.

```
2907 \cs_new_protected:Npn \@_make_m_preamble:n #1
2908 {
2909   \str_case:nnF { #1 }
2910   {
2911     c { \@_make_m_preamble_i:n #1 }
2912     l { \@_make_m_preamble_i:n #1 }
2913     r { \@_make_m_preamble_i:n #1 }
2914     > { \@_make_m_preamble_ii:nn #1 }
2915     ! { \@_make_m_preamble_ii:nn #1 }
2916     @ { \@_make_m_preamble_ii:nn #1 }
2917     | { \@_make_m_preamble_iii:n #1 }
2918     p { \@_make_m_preamble_iv:nnn t #1 }
2919     m { \@_make_m_preamble_iv:nnn c #1 }
2920     b { \@_make_m_preamble_iv:nnn b #1 }
2921     w { \@_make_m_preamble_v:nnnn { } #1 }
2922     W { \@_make_m_preamble_v:nnnn { \@_special_W: } #1 }
2923     \q_stop { }
2924   }
2925   {
2926     \cs_if_exist:cTF { NC @ find @ #1 }
2927     {
2928       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2929       \exp_last_unbraced:No \@_make_m_preamble:n \l_tmpa_tl
2930     }
2931     {
2932       \str_if_eq:nnTF { #1 } { S }
2933       { \@_fatal:n { unknown~column~type~S~multicolumn } }
2934       { \@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
2935     }
2936   }
2937 }
```

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

```
2938 \cs_new_protected:Npn \@_make_m_preamble_i:n #1
2939 {
2940   \tl_gput_right:Nn \g_@@_preamble_tl
2941   {
2942     > { \@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
2943     #1
2944     < \@_cell_end:
2945   }
```

We test for the presence of a `<`.

```
2946 \@_make_m_preamble_x:n
2947 }
```

For >, ! and @

```

2948 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2949 {
2950   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2951   \@@_make_m_preamble:n
2952 }

```

For |

```

2953 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2954 {
2955   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2956   \@@_make_m_preamble:n
2957 }

```

For p, m and b

```

2958 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2959 {
2960   \tl_gput_right:Nn \g_@@_preamble_tl
2961   {
2962     > {
2963       \@@_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`.

```

2964       \setlength { \l_tmpa_dim } { #3 }
2965       \begin { minipage } [ #1 ] { \l_tmpa_dim }
2966       \mode_leave_vertical:
2967       \arraybackslash
2968       \vrule height \box_ht:N \@@arstrutbox depth \c_zero_dim width \c_zero_dim
2969     }
2970     c
2971     < {
2972       \vrule height \c_zero_dim depth \box_dp:N \@@arstrutbox width \c_zero_dim
2973       \end { minipage }
2974       \@@_cell_end:
2975     }
2976   }

```

We test for the presence of a <.

```

2977   \@@_make_m_preamble_x:n
2978 }

```

For w and W

```

2979 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2980 {
2981   \tl_gput_right:Nn \g_@@_preamble_tl
2982   {
2983     > {
2984       \dim_set:Nn \l_@@_col_width_dim { #4 }
2985       \hbox_set:Nw \l_@@_cell_box
2986       \@@_cell_begin:
2987       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2988     }
2989     c
2990     < {
2991       \@@_cell_end:
2992       \hbox_set_end:
2993       \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
2994       #1
2995       \@@_adjust_size_box:
2996       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2997     }
2998   }

```

We test for the presence of a <.

```
2999 \@@_make_m_preamble_x:n
3000 }
```

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

```
3001 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3002 {
3003   \str_if_eq:nnTF { #1 } { < }
3004     { \@@_make_m_preamble_ix:n }
3005     { \@@_make_m_preamble:n { #1 } }
3006 }
3007 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3008 {
3009   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3010   \@@_make_m_preamble_x:n
3011 }
```

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`).

```
3012 \cs_new_protected:Npn \@@_put_box_in_flow:
3013 {
3014   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3015   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3016   \str_if_eq:eeTF \l_@@_baseline_tl { c }
3017     { \box_use_drop:N \l_tmpa_box }
3018     { \@@_put_box_in_flow_i: }
3019 }
```

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

```
3020 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3021 {
3022   \pgfpicture
3023     \@@_qpoint:n { row - 1 }
3024     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3025     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3026     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3027     \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).

```
3028   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3029   {
3030     \int_set:Nn \l_tmpa_int
3031       {
3032         \str_range:Nnn
3033           \l_@@_baseline_tl
3034           6
3035           { \tl_count:o \l_@@_baseline_tl }
3036       }
3037     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3038   }
3039   {
3040     \str_if_eq:eeTF \l_@@_baseline_tl { t }
3041     { \int_set_eq:NN \l_tmpa_int \c_one_int }
3042     {
3043       \str_if_eq:onTF \l_@@_baseline_tl { b }
3044       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3045       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
```

```

3046     }
3047     \bool_lazy_or:nnT
3048     { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3049     { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3050     {
3051         \@@_error:n { bad-value-for-baseline }
3052         \int_set_eq:NN \l_tmpa_int \c_one_int
3053     }
3054     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3055     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3056     }
3057     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3058     \endpgfpicture
3059     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3060     \box_use_drop:N \l_tmpa_box
3061 }

```

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).

```

3062 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3063 {

```

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.

```

3064     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3065     {
3066         \int_compare:nNnT { \c@jCol } > { \c_one_int }
3067         {
3068             \box_set_wd:Nn \l_@@_the_array_box
3069             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3070         }
3071     }

```

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

```

3072     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3073     \bool_if:NT \l_@@_caption_above_bool
3074     {
3075         \tl_if_empty:NF \l_@@_caption_tl
3076         {
3077             \bool_set_false:N \g_@@_caption_finished_bool
3078             \int_gzero:N \c@tabularnote
3079             \@@_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).

```

3080         \int_compare:nNnT { \g_@@_notes_caption_int } > { \c_zero_int }
3081         {
3082             \tl_gput_right:Ne \g_@@_aux_tl
3083             {
3084                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3085                 { \int_use:N \g_@@_notes_caption_int }
3086             }
3087             \int_gzero:N \g_@@_notes_caption_int
3088         }
3089     }
3090 }

```


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

```

3091   \hbox
3092   {
3093     \box_use_drop:N \l_@@_the_array_box

```

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

```

3094     \@@_create_extra_nodes:
3095     \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3096   }

```

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).

```

3097   \bool_lazy_any:nT
3098   {
3099     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3100     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3101     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3102   }
3103   \@@_insert_tabularnotes:
3104   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3105   \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3106   \end { minipage }
3107 }

```

```

3108 \cs_new_protected:Npn \@@_insert_caption:
3109 {
3110   \tl_if_empty:NF \l_@@_caption_tl
3111   {
3112     \cs_if_exist:NTF \capttype
3113     { \@@_insert_caption_i: }
3114     { \@@_error:n { caption~outside~float } }
3115   }
3116 }

```

```

3117 \cs_new_protected:Npn \@@_insert_caption_i:
3118 {
3119   \group_begin:

```

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

```

3120   \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.

```

3121   \IfPackageLoadedT { floatrow }
3122   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3123   \tl_if_empty:NTF \l_@@_short_caption_tl
3124   { \caption }
3125   { \caption [ \l_@@_short_caption_tl ] }
3126   { \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.

```

3127 \bool_if:NF \g_@@_caption_finished_bool
3128 {
3129     \bool_gset_true:N \g_@@_caption_finished_bool
3130     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3131     \int_gzero:N \c@tabularnote
3132 }
3133 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3134 \group_end:
3135 }
3136 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3137 {
3138     \@@_error_or_warning:n { tabularnote-below-the-tabular }
3139     \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3140 }
3141 \cs_new_protected:Npn \@@_insert_tabularnotes:
3142 {
3143     \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3144     \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3145     \skip_vertical:N 0.65ex

```

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

```

3146 \group_begin:
3147 \l_@@_notes_code_before_tl
3148 \tl_if_empty:NF \g_@@_tabularnote_tl
3149 {
3150     \g_@@_tabularnote_tl \par
3151     \tl_gclear:N \g_@@_tabularnote_tl
3152 }

```

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.

```

3153 \int_compare:nNnT { \c@tabularnote } > { \c_zero_int }
3154 {
3155     \bool_if:NTF \l_@@_notes_para_bool
3156     {
3157         \begin { tabularnotes* }
3158         \seq_map_inline:Nn \g_@@_notes_seq
3159             { \@@_one_tabularnote:nn ##1 }
3160         \strut
3161         \end { tabularnotes* }

```

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

```

3162 \par
3163 }
3164 {
3165     \tabularnotes
3166     \seq_map_inline:Nn \g_@@_notes_seq
3167         { \@@_one_tabularnote:nn ##1 }
3168     \strut
3169     \endtabularnotes
3170 }
3171 }
3172 \unskip
3173 \group_end:
3174 \bool_if:NT \l_@@_notes_bottomrule_bool
3175 {
3176     \IfPackageLoadedTF { booktabs }
3177     {

```

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

```

3178 \skip_vertical:N \aboverulesep

```

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

```

3179         { \CT@arc@ \hrule height \heavyrulewidth }
3180     }
3181     { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3182 }
3183 \l_@@_notes_code_after_tl
3184 \seq_gclear:N \g_@@_notes_seq
3185 \seq_gclear:N \g_@@_notes_in_caption_seq
3186 \int_gzero:N \c@tabularnote
3187 }

```

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

```

3188 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3189 {
3190     \tl_if_novalue:nTF { #1 }
3191     { \item }
3192     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3193 }

```

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.

```

3194 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3195 {
3196     \pgfpicture
3197     \@@_qpoint:n { row - 1 }
3198     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3199     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3200     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3201     \endpgfpicture
3202     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3203     \int_if_zero:nT { \l_@@_first_row_int }
3204     {
3205         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3206         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3207     }
3208     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3209 }

```

Now, the general case.

```

3210 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3211 {

```

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

```

3212     \str_if_eq:eeT \l_@@_baseline_tl { t }
3213     { \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`.

```

3214     \pgfpicture
3215     \@@_qpoint:n { row - 1 }
3216     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3217     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3218     {
3219         \int_set:Nn \l_tmpa_int
3220         {
3221             \str_range:Nnn
3222             \l_@@_baseline_tl
3223             { 6 }
3224             { \tl_count:o \l_@@_baseline_tl }

```

```

3225     }
3226     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3227 }
3228 {
3229     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3230     \bool_lazy_or:nnT
3231     { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3232     { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3233     {
3234         \@@_error:n { bad-value-for~baseline }
3235         \int_set:Nn \l_tmpa_int 1
3236     }
3237     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3238 }
3239 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3240 \endpgfpicture
3241 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3242 \int_if_zero:nT { \l_@@_first_row_int }
3243 {
3244     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3245     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3246 }
3247 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3248 }

```

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.

```

3249 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3250 {

```

We will compute the real width of both delimiters used.

```

3251     \dim_zero_new:N \l_@@_real_left_delim_dim
3252     \dim_zero_new:N \l_@@_real_right_delim_dim
3253     \hbox_set:Nn \l_tmpb_box
3254     {
3255         \m@th % added 2024/11/21
3256         \c_math_toggle_token
3257         \left #1
3258         \vcenter
3259         {
3260             \vbox_to_ht:nn
3261             { \box_ht_plus_dp:N \l_tmpa_box }
3262             { }
3263         }
3264         \right .
3265         \c_math_toggle_token
3266     }
3267     \dim_set:Nn \l_@@_real_left_delim_dim
3268     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3269     \hbox_set:Nn \l_tmpb_box
3270     {
3271         \m@th % added 2024/11/21
3272         \c_math_toggle_token
3273         \left .
3274         \vbox_to_ht:nn
3275         { \box_ht_plus_dp:N \l_tmpa_box }
3276         { }
3277         \right #2
3278         \c_math_toggle_token
3279     }
3280     \dim_set:Nn \l_@@_real_right_delim_dim
3281     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3282     \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3283     \@@_put_box_in_flow:
3284     \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3285 }

```

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).

```

3286 \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).

```

3287 {
3288   \peek_remove_spaces:n
3289   {
3290     \peek_meaning:NTF \end
3291     { \@@_analyze_end:Nn }
3292     {
3293       \@@_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`).

```

3294       \@@_array:o \g_@@_array_preamble_tl
3295     }
3296   }
3297 }
3298 {
3299   \@@_create_col_nodes:
3300   \endarray
3301 }

```

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

```

3302 \NewDocumentEnvironment { @@-light-syntax } { b }
3303 {

```

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`.

```

3304   \tl_if_empty:nT { #1 }
3305   { \@@_fatal:n { empty-environment } }
3306   \tl_if_in:nnT { #1 } { & }
3307   { \@@_fatal:n { ampersand-in-light-syntax } }
3308   \tl_if_in:nnT { #1 } { \ }
3309   { \@@_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`.

```

3310   \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3311 }

```

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.

```

3312 {
3313   \@@_create_col_nodes:
3314   \endarray
3315 }

3316 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3317 {
3318   \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).

```

3319   \seq_clear_new:N \l_@@_rows_seq

```

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

```

3320   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3321   \bool_if:NTF \l_@@_light_syntax_expanded_bool
3322     { \seq_set_split:Nee }
3323     { \seq_set_split:Non }
3324   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3325   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3326   \tl_if_empty:NF \l_tmpa_tl
3327   { \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.

```

3328   \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
3329     { \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`).

```

3330   \tl_build_begin:N \l_@@_new_body_tl
3331   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3332   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3333   \@@_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).

```

3334   \seq_map_inline:Nn \l_@@_rows_seq
3335   {
3336     \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3337     \@@_line_with_light_syntax:n { ##1 }
3338   }
3339   \tl_build_end:N \l_@@_new_body_tl

3340   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
3341   {
3342     \int_set:Nn \l_@@_last_col_int
3343       { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3344   }

```

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.

```

3345   \@@_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`).

```

3346   \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3347 }

```

```

3348 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3349 {
3350   \seq_clear_new:N \l_@@_cells_seq
3351   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3352   \int_set:Nn \l_@@_nb_cols_int
3353   {
3354     \int_max:nn
3355     { \l_@@_nb_cols_int }
3356     { \seq_count:N \l_@@_cells_seq }
3357   }
3358   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3359   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3360   \seq_map_inline:Nn \l_@@_cells_seq
3361   { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3362 }
3363 \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`.

```

3364 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3365 {
3366   \str_if_eq:eeT \g_@@_name_env_str { #2 }
3367   { \@@_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.

```

3368   \end { #2 }
3369 }

```

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`).

```

3370 \cs_new:Npn \@@_create_col_nodes:
3371 {
3372   \crrr
3373   \int_if_zero:nT { \l_@@_first_col_int }
3374   {
3375     \omit
3376     \hbox_overlap_left:n
3377     {
3378       \bool_if:NT \l_@@_code_before_bool
3379       { \pgfsys@markposition { \@@_env: - col - 0 } }
3380       \pgfpicture
3381       \pgfrememberpicturepositiononpagetrue
3382       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3383       \str_if_empty:NF \l_@@_name_str
3384       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3385       \endpgfpicture
3386       \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3387     }
3388     &
3389   }
3390   \omit

```

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

```

3391   \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`).

```

3392   \int_if_zero:nTF { \l_@@_first_col_int }
3393   {
3394     \@@_mark_position:n { 1 }

```

```

3395 \pgfpicture
3396 \pgfrememberpicturepositiononpagetrue
3397 \pgfcoordinate { \@@_env: - col - 1 }
3398 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3399 \str_if_empty:NF \l_@@_name_str
3400 { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3401 \endpgfpicture
3402 }
3403 {
3404 \bool_if:NT \l_@@_code_before_bool
3405 {
3406 \hbox
3407 {
3408 \skip_horizontal:n { 0.5 \arrayrulewidth }
3409 \pgfsys@markposition { \@@_env: - col - 1 }
3410 \skip_horizontal:n { -0.5 \arrayrulewidth }
3411 }
3412 }
3413 \pgfpicture
3414 \pgfrememberpicturepositiononpagetrue
3415 \pgfcoordinate { \@@_env: - col - 1 }
3416 { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3417 \@@_node_alias:n { 1 }
3418 \endpgfpicture
3419 }

```

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.

```

3420 \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
3421 \bool_if:NF \l_@@_auto_columns_width_bool
3422 { \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim } }
3423 {
3424 \bool_lazy_and:nnTF
3425 { \l_@@_auto_columns_width_bool }
3426 { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3427 { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3428 { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3429 \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3430 }
3431 \skip_horizontal:N \g_tmpa_skip
3432 \hbox
3433 {
3434 \@@_mark_position:n { 2 }
3435 \pgfpicture
3436 \pgfrememberpicturepositiononpagetrue
3437 \pgfcoordinate { \@@_env: - col - 2 }
3438 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3439 \@@_node_alias:n { 2 }
3440 \endpgfpicture
3441 }

```

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.

```

3442 \int_gset_eq:NN \g_tmpa_int \c_one_int
3443 \bool_if:NTF \g_@@_last_col_found_bool
3444 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3445 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3446 {
3447 &
3448 \omit

```



```
3449 \int_gincr:N \g_tmpa_int
```

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

```
3450 \skip_horizontal:N \g_tmpa_skip
3451 \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }
```

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

```
3452 \pgfpicture
3453 \pgfrememberpicturepositiononpagetrue
3454 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3455 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3456 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3457 \endpgfpicture
3458 }
```

```
3459 &
3460 \omit
```

If there is only one column (and a potential “last column”), we don’t have to put the following code (there is only one column and we have put the correct code previously).

```
3461 \bool_lazy_or:nnF
3462 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3463 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3464 {
3465   \skip_horizontal:N \g_tmpa_skip
3466   \int_gincr:N \g_tmpa_int
3467   \bool_lazy_any:nF
3468   {
3469     \g_@@_delims_bool
3470     \l_@@_tabular_bool
3471     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3472     \l_@@_exterior_arraycolsep_bool
3473     \l_@@_bar_at_end_of_pream_bool
3474   }
3475   { \skip_horizontal:n { - \col@sep } }
3476   \bool_if:NT \l_@@_code_before_bool
3477   {
3478     \hbox
3479     {
3480       \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.

```
3481 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3482 { \skip_horizontal:n { - \arraycolsep } }
3483 \pgfsys@markposition
3484 { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3485 \skip_horizontal:n { 0.5 \arrayrulewidth }
3486 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3487 { \skip_horizontal:N \arraycolsep }
3488 }
3489 }
3490 \pgfpicture
3491 \pgfrememberpicturepositiononpagetrue
3492 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3493 {
3494   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3495   {
3496     \pgfpoint
3497     { - 0.5 \arrayrulewidth - \arraycolsep }
3498     \c_zero_dim
3499   }
3500   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }

```

```

3501         }
3502         \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3503     \endpgfpicture
3504 }

3505 \bool_if:NT \g_@@_last_col_found_bool
3506 {
3507     \hbox_overlap_right:n
3508     {
3509         \skip_horizontal:N \g_@@_width_last_col_dim
3510         \skip_horizontal:N \col@sep
3511         \bool_if:NT \l_@@_code_before_bool
3512         {
3513             \pgfsys@markposition
3514             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3515         }
3516         \pgfpicture
3517         \pgfrememberpicturepositiononpagetrue
3518         \pgfcoordinate
3519         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3520         \pgfpointorigin
3521         \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3522         \endpgfpicture
3523     }
3524 }
3525 % \cr
3526 }

3527 \cs_new_protected:Npn \@@_mark_position:n #1
3528 {
3529     \bool_if:NT \l_@@_code_before_bool
3530     {
3531         \hbox
3532         {
3533             \skip_horizontal:n { -0.5 \arrayrulewidth }
3534             \pgfsys@markposition { \@@_env: - col - #1 }
3535             \skip_horizontal:n { 0.5 \arrayrulewidth }
3536         }
3537     }
3538 }

3539 \cs_new_protected:Npn \@@_node_alias:n #1
3540 {
3541     \str_if_empty:NF \l_@@_name_str
3542     { \pgfnodealias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3543 }

```

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

```

3544 \tl_const:Nn \c_@@_preamble_first_col_tl
3545 {
3546     >
3547     {

```

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

```

3548     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3549     \bool_gset_true:N \g_@@_after_col_zero_bool
3550     \@@_begin_of_row:
3551     \hbox_set:Nw \l_@@_cell_box
3552     \@@_math_toggle:
3553     \@@_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”.

```

3554     \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3555     {
3556         \bool_lazy_or:nnT
3557         { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3558         { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3559         {
3560             \l_@@_code_for_first_col_tl
3561             \xglobal \colorlet { nicematrix-first-col } { . }
3562         }
3563     }
3564 }

```

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`.

```

3565     l
3566     <
3567     {
3568         \@@_math_toggle:
3569         \hbox_set_end:
3570         \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3571         \@@_adjust_size_box:
3572         \@@_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.

```

3573     \dim_gset:Nn \g_@@_width_first_col_dim
3574     { \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.

```

3575     \hbox_overlap_left:n
3576     {
3577         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3578         { \@@_node_cell: }
3579         { \box_use_drop:N \l_@@_cell_box }
3580         \skip_horizontal:N \l_@@_left_delim_dim
3581         \skip_horizontal:N \l_@@_left_margin_dim
3582         \skip_horizontal:N \l_@@_extra_left_margin_dim
3583     }
3584     \bool_gset_false:N \g_@@_empty_cell_bool
3585     \skip_horizontal:n { -2 \col@sep }
3586 }
3587 }

```

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

```

3588 \tl_const:Nn \c_@@_preamble_last_col_tl
3589 {
3590     >
3591     {
3592         \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).

```

3593     \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.

```

3594     \bool_gset_true:N \g_@@_last_col_found_bool
3595     \int_gincr:N \c@jCol
3596     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3597     \hbox_set:Nw \l_@@_cell_box
3598     \@@_math_toggle:
3599     \@@_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”.

```

3600     \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3601     {
3602         \bool_lazy_or:nnT
3603         { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3604         { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3605         {
3606             \l_@@_code_for_last_col_tl
3607             \xglobal \colorlet { nicematrix-last-col } { . }
3608         }
3609     }
3610 }
3611 1
3612 <
3613 {
3614     \@@_math_toggle:
3615     \hbox_set_end:
3616     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3617     \@@_adjust_size_box:
3618     \@@_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.

```

3619     \dim_gset:Nn \g_@@_width_last_col_dim
3620     { \dim_max:nn { \g_@@_width_last_col_dim } { \box_wd:N \l_@@_cell_box } }
3621     \skip_horizontal:n { -2 \col@sep }

```

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

```

3622     \hbox_overlap_right:n
3623     {
3624         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3625         {
3626             \skip_horizontal:N \l_@@_right_delim_dim
3627             \skip_horizontal:N \l_@@_right_margin_dim
3628             \skip_horizontal:N \l_@@_extra_right_margin_dim
3629             \@@_node_cell:
3630         }
3631     }
3632     \bool_gset_false:N \g_@@_empty_cell_bool
3633 }
3634 }

```

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

```

3635 \NewDocumentEnvironment { NiceArray } { }
3636 {
3637     \bool_gset_false:N \g_@@_delims_bool
3638     \str_if_empty:NT \g_@@_name_env_str
3639     { \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).

```

3640     \NiceArrayWithDelims . .
3641 }
3642 { \endNiceArrayWithDelims }

```

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

```

3643 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3644 {
3645     \NewDocumentEnvironment { #1 NiceArray } { }
3646     {

```

```

3647 \bool_gset_true:N \g_@@_delims_bool
3648 \str_if_empty:NT \g_@@_name_env_str
3649 { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3650 \@@_test_if_math_mode:
3651 \NiceArrayWithDelims #2 #3
3652 }
3653 { \endNiceArrayWithDelims }
3654 }
3655 \@@_def_env:NNN p ( )
3656 \@@_def_env:NNN b [ ]
3657 \@@_def_env:NNN B \{ \}
3658 \@@_def_env:NNN v \vert \vert
3659 \@@_def_env:NNN V \Vert \Vert

```

13 The environment `{NiceMatrix}` and its variants

```

3660 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3661 {
3662   \bool_set_false:N \l_@@_preamble_bool
3663   \tl_clear:N \l_tmpa_tl
3664   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3665   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3666   \tl_put_right:Nn \l_tmpa_tl
3667   {
3668     *
3669     {
3670       \int_case:nnF \l_@@_last_col_int
3671       {
3672         { -2 } { \c@MaxMatrixCols }
3673         { -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).

```

3674   }
3675   { \int_eval:n { \l_@@_last_col_int - 1 } }
3676 }
3677 { #2 }
3678 }
3679 \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3680 \exp_args:No \l_tmpb_tl \l_tmpa_tl
3681 }
3682 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3683 \clist_map_inline:nn { p , b , B , v , V }
3684 {
3685   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3686   {
3687     \bool_gset_true:N \g_@@_delims_bool
3688     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3689     \int_if_zero:nT { \l_@@_last_col_int }
3690     {
3691       \bool_set_true:N \l_@@_last_col_without_value_bool
3692       \int_set:Nn \l_@@_last_col_int { -1 }
3693     }
3694     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3695     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3696   }
3697   { \use:c { end #1 NiceArray } }
3698 }

```

We define also an environment `{NiceMatrix}`

```

3699 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3700 {
3701   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3702   \int_if_zero:nT { \l_@@_last_col_int }
3703   {
3704     \bool_set_true:N \l_@@_last_col_without_value_bool
3705     \int_set:Nn \l_@@_last_col_int { -1 }
3706   }
3707   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3708   \bool_lazy_or:nnT
3709     { \clist_if_empty_p:N \l_@@_vlines_clist }
3710     { \l_@@_except_borders_bool }
3711     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3712   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3713 }
3714 { \endNiceArray }

```

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

```

3715 \cs_new_protected:Npn \@@_NotEmpty:
3716 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3717 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3718 {

```

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.

```

3719   \dim_compare:nNnT { \l_@@_width_dim } = { \c_zero_dim }
3720     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3721   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3722   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3723   \tl_if_empty:NF \l_@@_short_caption_tl
3724   {
3725     \tl_if_empty:NT \l_@@_caption_tl
3726     {
3727       \@@_error_or_warning:n { short-caption-without-caption }
3728       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3729     }
3730   }
3731   \tl_if_empty:NF \l_@@_label_tl
3732   {
3733     \tl_if_empty:NT \l_@@_caption_tl
3734     { \@@_error_or_warning:n { label-without-caption } }
3735   }
3736   \NewDocumentEnvironment { TabularNote } { b }
3737   {
3738     \bool_if:NTF \l_@@_in_code_after_bool
3739       { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3740       {
3741         \tl_if_empty:NF \g_@@_tabularnote_tl
3742         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3743         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3744       }
3745     }
3746   { }
3747   \@@_settings_for_tabular:
3748   \NiceArray { #2 }
3749 }
3750 { \endNiceArray }
3751 \cs_new_protected:Npn \@@_settings_for_tabular:
3752 {

```

```

3753 \bool_set_true:N \l_@@_tabular_bool
3754 \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3755 \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3756 \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3757 }

3758 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3759 {
3760 \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3761 \dim_set:Nn \l_@@_width_dim { #1 }
3762 \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3763 \@@_settings_for_tabular:
3764 \NiceArray { #3 }
3765 }
3766 {
3767 \endNiceArray
3768 \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3769 { \@@_error:n { NiceTabularX~without~X } }
3770 }

3771 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3772 {
3773 \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3774 \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3775 \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3776 \@@_settings_for_tabular:
3777 \NiceArray { #3 }
3778 }
3779 { \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`).

```

3780 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3781 {
3782 \bool_lazy_all:nT
3783 {
3784 { \dim_compare_p:nNn { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim } }
3785 { \l_@@_hvlines_bool }
3786 { ! \g_@@_delims_bool }
3787 { ! \l_@@_except_borders_bool }
3788 }
3789 {
3790 \bool_set_true:N \l_@@_except_borders_bool
3791 \clist_if_empty:NF \l_@@_corners_clist
3792 { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3793 \tl_gput_right:Nn \g_@@_pre_code_after_tl
3794 {
3795 \@@_stroke_block:nnn
3796 {
3797 rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3798 draw = \l_@@_rules_color_tl
3799 }
3800 { 1-1 }
3801 { \int_use:N \c@iRow - \int_use:N \c@jCol }
3802 }
3803 }
3804 }

```

```

3805 \cs_new_protected:Npn \@@_after_array:
3806 {

```

There was a `\hook_gput_code:nnn { env / tabular / begin } { nicematrix }` in the command `\@@_pre_array_ii:` in order to come back to the standard definition of `\multicolumn` (in the tabulars used by the final user in the cells of our array of `nicematrix`) and maybe another linked to `colortbl`.

```

3807 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3808 \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.

```

3809 \bool_if:NT \g_@@_last_col_found_bool
3810 { \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`.

```

3811 \bool_if:NT \l_@@_last_col_without_value_bool
3812 { \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.

```

3813 \bool_if:NT \l_@@_last_row_without_value_bool
3814 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3815 \tl_gput_right:Ne \g_@@_aux_tl
3816 {
3817   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3818   {
3819     \int_use:N \l_@@_first_row_int ,
3820     \int_use:N \c@iRow ,
3821     \int_use:N \g_@@_row_total_int ,
3822     \int_use:N \l_@@_first_col_int ,
3823     \int_use:N \c@jCol ,
3824     \int_use:N \g_@@_col_total_int
3825   }
3826 }

```

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`).

```

3827 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3828 {
3829   \tl_gput_right:Ne \g_@@_aux_tl
3830   {
3831     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3832     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3833   }
3834 }
3835 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3836 {
3837   \tl_gput_right:Ne \g_@@_aux_tl
3838   {
3839     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3840     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3841     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3842     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3843   }
3844 }

```


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

```
3845 \@@_create_diag_nodes:
```

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

```
3846 \pgfpicture
3847 \@@_create_aliases_last:
3848 \str_if_empty:NF \l_@@_name_str { \@@_create_alias_nodes: }
3849 \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.

```
3850 \bool_if:NT \l_@@_parallelize_diags_bool
3851 {
3852   \int_gzero:N \g_@@_ddots_int
3853   \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.

```
3854   \dim_gzero:N \g_@@_delta_x_one_dim
3855   \dim_gzero:N \g_@@_delta_y_one_dim
3856   \dim_gzero:N \g_@@_delta_x_two_dim
3857   \dim_gzero:N \g_@@_delta_y_two_dim
3858 }
3859 \bool_set_false:N \l_@@_initial_open_bool
3860 \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.

```
3861 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }
```

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

```
3862 \@@_draw_dotted_lines:
```

The following computes the “corners” (made up of empty cells) but if there is no corner to compute, it won’t do anything. The corners are computed in `\l_@@_corners_cells_clist` which will contain all the cells which are empty (and not in a block) considered in the corners of the array.

```
3863 \clist_if_empty:NF \l_@@_corners_clist
3864 {
3865   \bool_if:NTF \l_@@_no_cell_nodes_bool
3866   { \@@_error:n { corners~with~no~cell~nodes } }
3867   { \@@_compute_corners: }
3868 }
```

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

```
3869 \@@_adjust_pos_of_blocks_seq:
3870 \@@_deal_with_rounded_corners:
3871 \clist_if_empty:NF \l_@@_hlines_clist { \@@_draw_hlines: }
3872 \clist_if_empty:NF \l_@@_vlines_clist { \@@_draw_vlines: }
```

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

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

```

3873 \IfPackageLoadedT { tikz }
3874 {
3875   \tikzset
3876   {
3877     every-picture / .style =
3878     {
3879       overlay ,
3880       remember-picture ,
3881       name-prefix = \@@_env: -
3882     }
3883   }
3884 }
3885 \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
3886 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3887 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3888 \cs_set_eq:NN \OverBrace \@@_OverBrace
3889 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3890 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3891 \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.

```

3892 \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
3893 \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.

```

3894 \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).

```

3895 \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`).

```

3896 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3897 { \@@_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:`.

```

3898 \bool_set_true:N \l_@@_in_code_after_bool
3899 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3900 \scan_stop:
3901 \tl_gclear:N \g_nicematrix_code_after_tl
3902 \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.

```

3903 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3904 \tl_if_empty:NF \g_@@_pre_code_before_tl
3905 {
3906   \tl_gput_right:Ne \g_@@_aux_tl
3907   {
3908     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3909     { \exp_not:o \g_@@_pre_code_before_tl }
3910   }
3911   \tl_gclear:N \g_@@_pre_code_before_tl
3912 }

```

```

3913 \tl_if_empty:NF \g_nicematrix_code_before_tl
3914 {
3915   \tl_gput_right:Ne \g_@@_aux_tl
3916   {
3917     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3918     { \exp_not:o \g_nicematrix_code_before_tl }
3919   }
3920   \tl_gclear:N \g_nicematrix_code_before_tl
3921 }

3922 \str_gclear:N \g_@@_name_env_str
3923 \@@_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`.

```

3924 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3925 }

3926 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
3927 {
3928   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3929   \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }

3930   \dim_set:Nn \l_@@_xdots_shorten_start_dim
3931   { 0.6 \l_@@_xdots_shorten_start_dim }
3932   \dim_set:Nn \l_@@_xdots_shorten_end_dim
3933   { 0.6 \l_@@_xdots_shorten_end_dim }
3934 }

```

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

The following command will extract the potential options (between square brackets) at the beginning of the `\CodeAfter` (that is to say, when `\CodeAfter` is used, the options of that “command” `\CodeAfter`). Idem for the `\CodeBefore`.

```

3935 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3936 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

3937 \cs_new_protected:Npn \@@_create_alias_nodes:
3938 {
3939   \int_step_inline:nn { \c{iRow }
3940     {
3941       \pgfnodealias
3942       { \l_@@_name_str - ##1 - last }
3943       { \@@_env: - ##1 - \int_use:N \c{jCol }
3944     }
3945   \int_step_inline:nn { \c{jCol }
3946     {
3947       \pgfnodealias
3948       { \l_@@_name_str - last - ##1 }
3949       { \@@_env: - \int_use:N \c{iRow } - ##1 }
3950     }
3951   \pgfnodealias % added 2025-04-05
3952   { \l_@@_name_str - last - last }
3953   { \@@_env: - \int_use:N \c{iRow } - \int_use:N \c{jCol }
3954 }

```

¹³e.g. `\color[rgb]{0.5,0.5,0}`

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format $i-j$. However, the user is allowed to omit i or j (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_pos_of_blocks_seq` (and `\g_@@_blocks_seq`) as a number of rows (resp. columns) for the block equal to 100. It's possible, after the construction of the array, to replace these values by the correct ones (since we know the number of rows and columns of the array).

```

3955 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3956 {
3957   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3958   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3959 }

```

The following command must *not* be protected.

```

3960 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3961 {
3962   { #1 }
3963   { #2 }
3964   {
3965     \int_compare:nNnTF { #3 } > { 98 }
3966     { \int_use:N \c@iRow }
3967     { #3 }
3968   }
3969   {
3970     \int_compare:nNnTF { #4 } > { 98 }
3971     { \int_use:N \c@jCol }
3972     { #4 }
3973   }
3974   { #5 }
3975 }

```

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).

```

3976 \hook_gput_code:nnn { begindocument } { . }
3977 {
3978   \cs_new_protected:Npe \@@_draw_dotted_lines:
3979   {
3980     \c_@@_pgfortikzpicture_tl
3981     \@@_draw_dotted_lines_i:
3982     \c_@@_endpgfortikzpicture_tl
3983   }
3984 }

```

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

```

3985 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3986 {
3987   \pgfrememberpicturepositiononpagetrue
3988   \pgf@relevantforpicturesizefalse
3989   \g_@@_HVdotsfor_lines_tl
3990   \g_@@_Vdots_lines_tl
3991   \g_@@_Ddots_lines_tl
3992   \g_@@_Idots_lines_tl
3993   \g_@@_Cdots_lines_tl
3994   \g_@@_Ldots_lines_tl
3995 }

3996 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3997 {
3998   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3999   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4000 }

```

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

```

4001 \pgfdeclareshape { @@_diag_node }
4002 {
4003   \savedanchor { \five }
4004   {
4005     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4006     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4007   }
4008   \anchor { 5 } { \five }
4009   \anchor { center } { \pgfpointorigin }
4010   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4011   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4012   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4013   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4014   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4015   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4016   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4017   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4018   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4019   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4020 }

```

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).

```

4021 \cs_new_protected:Npn \@@_create_diag_nodes:
4022 {
4023   \pgfpicture
4024   \pgfrememberpicturepositiononpagetrue
4025   \int_step_inline:nn { \int_max:nn { \c@iRow } { \c@jCol } }
4026   {
4027     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4028     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4029     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4030     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4031     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4032     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4033     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4034     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4035     \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.

```

4036     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4037     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4038     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4039     \str_if_empty:NF \l_@@_name_str
4040     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4041   }

```

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

```

4042   \int_set:Nn \l_tmpa_int { \int_max:nn { \c@iRow } { \c@jCol } + 1 }
4043   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4044   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4045   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4046   \pgfcoordinate
4047   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4048   \pgfnodealias
4049   { \@@_env: - last }
4050   { \@@_env: - \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
4051   \str_if_empty:NF \l_@@_name_str
4052   {

```

```

4053     \pgfnodealias
4054     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4055     { \@@_env: - \int_use:N \l_tmpa_int }
4056     \pgfnodealias
4057     { \l_@@_name_str - last }
4058     { \@@_env: - last }
4059   }
4060 \endpgfpicture
4061 }

```

16 We draw the dotted lines

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

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

The command `\@@_find_extremities_of_line:nnnn` takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the x -value of the orientation vector of the line;
- the fourth argument is the y -value of the orientation vector of the line.

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

```

4062 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4063 {

```

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

```

4064     \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4065     \int_set:Nn \l_@@_initial_i_int { #1 }
4066     \int_set:Nn \l_@@_initial_j_int { #2 }
4067     \int_set:Nn \l_@@_final_i_int { #1 }
4068     \int_set:Nn \l_@@_final_j_int { #2 }

```

We will do two loops: one when determining the initial cell and the other when determining the final cell. The boolean `\l_@@_stop_loop_bool` will be used to control these loops. In the first loop, we search the “final” extremity of the line.

```

4069     \bool_set_false:N \l_@@_stop_loop_bool
4070     \bool_do_until:Nn \l_@@_stop_loop_bool
4071     {
4072         \int_add:Nn \l_@@_final_i_int { #3 }
4073         \int_add:Nn \l_@@_final_j_int { #4 }
4074         \bool_set_false:N \l_@@_final_open_bool

```

We test if we are still in the matrix. Since this is the core of the loop, we **optimize** the code by using a TeX-style of conditionals.

```

4075     \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4076     \if_int_compare:w #3 = \c_one_int
4077     \bool_set_true:N \l_@@_final_open_bool
4078   \else:
4079     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4080     \bool_set_true:N \l_@@_final_open_bool
4081   \fi:
4082 \fi:
4083 \else:
4084   \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4085   \if_int_compare:w #4 = -1
4086   \bool_set_true:N \l_@@_final_open_bool
4087   \fi:
4088 \else:
4089   \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4090   \if_int_compare:w #4 = \c_one_int
4091   \bool_set_true:N \l_@@_final_open_bool
4092   \fi:
4093 \fi:
4094 \fi:
4095 \fi:
4096 \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.

```

4097   {

```

We do a step backwards.

```

4098     \int_sub:Nn \l_@@_final_i_int { #3 }
4099     \int_sub:Nn \l_@@_final_j_int { #4 }
4100     \bool_set_true:N \l_@@_stop_loop_bool
4101   }

```

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`.

```

4102   {
4103     \cs_if_exist:cTF
4104     {
4105       @@ _ dotted _
4106       \int_use:N \l_@@_final_i_int -
4107       \int_use:N \l_@@_final_j_int
4108     }
4109     {
4110       \int_sub:Nn \l_@@_final_i_int { #3 }
4111       \int_sub:Nn \l_@@_final_j_int { #4 }
4112       \bool_set_true:N \l_@@_final_open_bool
4113       \bool_set_true:N \l_@@_stop_loop_bool
4114     }
4115     {
4116       \cs_if_exist:cTF
4117       {
4118         pgf @ sh @ ns @ \@@_env:
4119         - \int_use:N \l_@@_final_i_int
4120         - \int_use:N \l_@@_final_j_int
4121       }
4122       { \bool_set_true:N \l_@@_stop_loop_bool }

```

If the case is empty, we declare that the cell as non-empty. Indeed, we will draw a dotted line and the cell will be on that dotted line. All the cells of a dotted line have to be marked as “dotted” because we don't want intersections between dotted lines. We recall that the research of the extremities of the lines are all done in the same TeX group (the group of the environment), even though, when the extremities are found, each line is drawn in a TeX group that we will open for the options of the line.

```

4123   {

```

```

4124         \cs_set_nopar:cpn
4125         {
4126             @@ _ dotted _
4127             \int_use:N \l_@@_final_i_int -
4128             \int_use:N \l_@@_final_j_int
4129         }
4130         { }
4131     }
4132 }
4133 }
4134 }

```

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

```

4135     \bool_set_false:N \l_@@_stop_loop_bool

```

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

```

4136     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4137     \bool_do_until:Nn \l_@@_stop_loop_bool
4138     {
4139         \int_sub:Nn \l_@@_initial_i_int { #3 }
4140         \int_sub:Nn \l_@@_initial_j_int { #4 }
4141         \bool_set_false:N \l_@@_initial_open_bool

```

We test if we are still in the matrix. Since this is the core of the loop, we **optimize** the code by using a TeX-style of conditionals.

```

4142         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4143         \if_int_compare:w #3 = \c_one_int
4144             \bool_set_true:N \l_@@_initial_open_bool
4145         \else:

```

`\l_tmpa_int` contains `\l_@@_col_min_int - 1` (only for efficiency).

```

4146         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4147             \bool_set_true:N \l_@@_initial_open_bool
4148         \fi:
4149     \fi:
4150 \else:
4151     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4152         \if_int_compare:w #4 = \c_one_int
4153             \bool_set_true:N \l_@@_initial_open_bool
4154         \fi:
4155     \else:
4156         \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4157             \if_int_compare:w #4 = -1
4158                 \bool_set_true:N \l_@@_initial_open_bool
4159             \fi:
4160         \fi:
4161     \fi:
4162 \fi:
4163 \bool_if:NTF \l_@@_initial_open_bool
4164 {
4165     \int_add:Nn \l_@@_initial_i_int { #3 }
4166     \int_add:Nn \l_@@_initial_j_int { #4 }
4167     \bool_set_true:N \l_@@_stop_loop_bool
4168 }
4169 {
4170     \cs_if_exist:cTF
4171     {
4172         @@ _ dotted _
4173         \int_use:N \l_@@_initial_i_int -
4174         \int_use:N \l_@@_initial_j_int
4175     }

```



```

4176     {
4177         \int_add:Nn \l_@@_initial_i_int { #3 }
4178         \int_add:Nn \l_@@_initial_j_int { #4 }
4179         \bool_set_true:N \l_@@_initial_open_bool
4180         \bool_set_true:N \l_@@_stop_loop_bool
4181     }
4182     {
4183         \cs_if_exist:cTF
4184         {
4185             pgf @ sh @ ns @ \@@_env:
4186             - \int_use:N \l_@@_initial_i_int
4187             - \int_use:N \l_@@_initial_j_int
4188         }
4189         { \bool_set_true:N \l_@@_stop_loop_bool }
4190     {
4191         \cs_set_nopar:cpn
4192         {
4193             @@ _ dotted _
4194             \int_use:N \l_@@_initial_i_int -
4195             \int_use:N \l_@@_initial_j_int
4196         }
4197         { }
4198     }
4199 }
4200 }
4201 }

```

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.

```

4202     \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4203     {
4204         { \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`.

```

4205         { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4206         { \int_use:N \l_@@_final_i_int }
4207         { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4208         { } % for the name of the block
4209     }
4210 }

```

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

```

4211 \cs_new_protected:Npn \@@_open_shorten:
4212 {
4213     \bool_if:NT \l_@@_initial_open_bool
4214     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4215     \bool_if:NT \l_@@_final_open_bool
4216     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4217 }

```

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).

```

4218 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4219 {
4220     \int_set_eq:NN \l_@@_row_min_int \c_one_int

```

```

4221 \int_set_eq:NN \l_@@_col_min_int \c_one_int
4222 \int_set_eq:NN \l_@@_row_max_int \c@iRow
4223 \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`.

```

4224 \seq_if_empty:NF \g_@@_submatrix_seq
4225 {
4226   \seq_map_inline:Nn \g_@@_submatrix_seq
4227   { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4228 }
4229 }

```

#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.

```

4230 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4231 {
4232   \if_int_compare:w #3 > #1
4233   \else:
4234     \if_int_compare:w #1 > #5
4235     \else:
4236       \if_int_compare:w #4 > #2
4237       \else:
4238         \if_int_compare:w #2 > #6
4239         \else:
4240           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4241           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4242           \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4243           \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4244         \fi:
4245       \fi:
4246     \fi:
4247   \fi:
4248 }

4249 \cs_new_protected:Npn \@@_set_initial_coords:
4250 {
4251   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4252   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4253 }
4254 \cs_new_protected:Npn \@@_set_final_coords:
4255 {

```

```

4256 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4257 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4258 }
4259 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4260 {
4261 \pgfpointanchor
4262 {
4263 \@@_env:
4264 - \int_use:N \l_@@_initial_i_int
4265 - \int_use:N \l_@@_initial_j_int
4266 }
4267 { #1 }
4268 \@@_set_initial_coords:
4269 }
4270 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4271 {
4272 \pgfpointanchor
4273 {
4274 \@@_env:
4275 - \int_use:N \l_@@_final_i_int
4276 - \int_use:N \l_@@_final_j_int
4277 }
4278 { #1 }
4279 \@@_set_final_coords:
4280 }
4281 \cs_new_protected:Npn \@@_open_x_initial_dim:
4282 {
4283 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4284 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4285 {
4286 \cs_if_exist:cT
4287 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4288 {
4289 \pgfpointanchor
4290 { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4291 { west }
4292 \dim_set:Nn \l_@@_x_initial_dim
4293 { \dim_min:nn { \l_@@_x_initial_dim } { \pgf@x } }
4294 }
4295 }

```

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

```

4296 \dim_compare:nNnT { \l_@@_x_initial_dim } = { \c_max_dim }
4297 {
4298 \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4299 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4300 \dim_add:Nn \l_@@_x_initial_dim \col@sep
4301 }
4302 }
4303 \cs_new_protected:Npn \@@_open_x_final_dim:
4304 {
4305 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4306 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4307 {
4308 \cs_if_exist:cT
4309 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4310 {
4311 \pgfpointanchor
4312 { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4313 { east }
4314 \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
4315 { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4316 }

```

```
4317     }
```

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

```
4318     \dim_compare:nNnT { \l_@@_x_final_dim } = { - \c_max_dim }
4319     {
4320         \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4321         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4322         \dim_sub:Nn \l_@@_x_final_dim \col@sep
4323     }
4324 }
```

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.

```
4325 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4326 {
4327     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4328     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4329     {
4330         \@@_find_extremities_of_line:nnnn { #1 } { #2 } 0 1
```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```
4331     \group_begin:
4332     \@@_open_shorten:
4333     \int_if_zero:nTF { #1 }
4334     { \color { nicematrix-first-row } }
4335     {
```

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.

```
4336         \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4337         { \color { nicematrix-last-row } }
4338     }
4339     \keys_set:nn { nicematrix / xdots } { #3 }
4340     \@@_color:o \l_@@_xdots_color_tl
4341     \@@_actually_draw_Ldots:
4342     \group_end:
4343 }
4344 }
```

The command `\@@_actually_draw_Ldots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

The following function is also used by `\Hdotsfor`.

```
4345 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4346 {
4347     \bool_if:NTF \l_@@_initial_open_bool
4348     {
4349         \@@_open_x_initial_dim:
4350         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4351         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4352     }
4353     { \@@_set_initial_coords_from_anchor:n { base-east } }
```

```

4354 \bool_if:NTF \l_@@_final_open_bool
4355 {
4356   \@@_open_x_final_dim:
4357   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4358   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4359 }
4360 { \@@_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.

```

4361 \bool_lazy_all:nTF
4362 {
4363   \l_@@_initial_open_bool
4364   \l_@@_final_open_bool
4365   { \int_compare_p:nNn { \l_@@_initial_i_int } = { \l_@@_last_row_int } }
4366 }
4367 {
4368   \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4369   \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4370 }

```

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 (?).

```

4371 {
4372   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4373   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4374 }
4375 \@@_draw_line:
4376 }

```

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.

```

4377 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4378 {
4379   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4380   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4381   {
4382     \@@_find_extremities_of_line:nnnn { #1 } { #2 } { 0 } { 1 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4383 \group_begin:
4384 \@@_open_shorten:
4385 \int_if_zero:NTF { #1 }
4386 { \color { nicematrix-first-row } }
4387 {

```

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.

```

4388 \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4389 { \color { nicematrix-last-row } }
4390 }
4391 \keys_set:nn { nicematrix / xdots } { #3 }
4392 \@@_color:o \l_@@_xdots_color_tl
4393 \@@_actually_draw_Cdots:
4394 \group_end:
4395 }
4396 }

```

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.

```

4397 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4398 {
4399   \bool_if:NTF \l_@@_initial_open_bool
4400     { \@@_open_x_initial_dim: }
4401     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4402   \bool_if:NTF \l_@@_final_open_bool
4403     { \@@_open_x_final_dim: }
4404     { \@@_set_final_coords_from_anchor:n { mid-west } }
4405   \bool_lazy_and:nnTF
4406     { \l_@@_initial_open_bool }
4407     { \l_@@_final_open_bool }
4408   {
4409     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4410     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4411     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4412     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4413     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4414   }
4415   {
4416     \bool_if:NT \l_@@_initial_open_bool
4417       { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4418     \bool_if:NT \l_@@_final_open_bool
4419       { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4420   }
4421   \@@_draw_line:
4422 }
4423 \cs_new_protected:Npn \@@_open_y_initial_dim:
4424 {
4425   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4426   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4427   {
4428     \cs_if_exist:cT
4429       { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4430     {
4431       \pgfpointanchor
4432         { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4433         { north }
4434       \dim_compare:nNnT { \pgf@y } > { \l_@@_y_initial_dim }
4435       { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4436     }
4437   }
4438   \dim_compare:nNnT { \l_@@_y_initial_dim } = { - \c_max_dim }
4439   {
4440     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4441     \dim_set:Nn \l_@@_y_initial_dim
4442     {
4443       \fp_to_dim:n
4444       {
4445         \pgf@y
4446         + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4447       }
4448     }
4449   }
4450 }

```

```

4451 \cs_new_protected:Npn \@@_open_y_final_dim:
4452 {
4453   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4454   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4455   {
4456     \cs_if_exist:cT
4457     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4458     {
4459       \pgfpointanchor
4460       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4461       { south }
4462       \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
4463       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4464     }
4465   }
4466   \dim_compare:nNnT { \l_@@_y_final_dim } = { \c_max_dim }
4467   {
4468     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4469     \dim_set:Nn \l_@@_y_final_dim
4470     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4471   }
4472 }

```

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.

```

4473 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4474 {
4475   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4476   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4477   {
4478     \@@_find_extremities_of_line:nnnn { #1 } { #2 } { 1 } { 0 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4479   \group_begin:
4480     \@@_open_shorten:
4481     \int_if_zero:nTF { #2 }
4482     { \color { nicematrix-first-col } }
4483     {
4484       \int_compare:nNnT { #2 } = { \l_@@_last_col_int }
4485       { \color { nicematrix-last-col } }
4486     }
4487     \keys_set:nn { nicematrix / xdots } { #3 }
4488     \@@_color:o \l_@@_xdots_color_tl
4489     \bool_if:NTF \l_@@_Vbrace_bool
4490     { \@@_actually_draw_Vbrace: }
4491     { \@@_actually_draw_Vdots: }
4492   \group_end:
4493 }
4494 }

```

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`.

```

4495 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4496 {
4497   \bool_lazy_and:nnTF { \l_@@_initial_open_bool } { \l_@@_final_open_bool }
4498   { \@@_actually_draw_Vdots_i: }
4499   { \@@_actually_draw_Vdots_ii: }
4500   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4501   \@@_draw_line:
4502 }

```

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

```

4503 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4504 {
4505   \@@_open_y_initial_dim:
4506   \@@_open_y_final_dim:
4507   \int_if_zero:nTF { \l_@@_initial_j_int }

```

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

```

4508   {
4509     \@@_qpoint:n { col - 1 }
4510     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4511     \dim_sub:Nn \l_@@_x_initial_dim
4512     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4513   }
4514   {
4515     \bool_lazy_and:nnTF
4516     { \int_compare_p:nNn { \l_@@_last_col_int } > { -2 } }
4517     {
4518       \int_compare_p:nNn
4519       { \l_@@_initial_j_int } = { \g_@@_col_total_int }
4520     }

```

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

```

4521   {
4522     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4523     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4524     \dim_add:Nn \l_@@_x_initial_dim
4525     { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4526   }

```

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

```

4527   {
4528     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4529     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4530     \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4531     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4532   }
4533 }
4534 }

```

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.

```

4535 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4536 {
4537   \bool_set_false:N \l_tmpa_bool
4538   \bool_if:NF \l_@@_initial_open_bool
4539   {
4540     \bool_if:NF \l_@@_final_open_bool
4541     {
4542       \@@_set_initial_coords_from_anchor:n { south-west }
4543       \@@_set_final_coords_from_anchor:n { north-west }
4544       \bool_set:Nn \l_tmpa_bool

```



```

4545         {
4546             \dim_compare_p:nNn
4547             { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4548         }
4549     }
4550 }

```

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

```

4551 \bool_if:NTF \l_@@_initial_open_bool
4552 {
4553     \@@_open_y_initial_dim:
4554     \@@_set_final_coords_from_anchor:n { north }
4555     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4556 }
4557 {
4558     \@@_set_initial_coords_from_anchor:n { south }
4559     \bool_if:NTF \l_@@_final_open_bool
4560     { \@@_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.

```

4561     {
4562         \@@_set_final_coords_from_anchor:n { north }
4563         \dim_compare:nNnF { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4564         {
4565             \dim_set:Nn \l_@@_x_initial_dim
4566             {
4567                 \bool_if:NTF \l_tmpa_bool { \dim_min:nn } { \dim_max:nn }
4568                 \l_@@_x_initial_dim \l_@@_x_final_dim
4569             }
4570         }
4571     }
4572 }
4573 }

```

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`.

```

4574 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4575 {
4576     \bool_if:NTF \l_@@_initial_open_bool
4577     { \@@_open_y_initial_dim: }
4578     { \@@_set_initial_coords_from_anchor:n { south } }
4579     \bool_if:NTF \l_@@_final_open_bool
4580     { \@@_open_y_final_dim: }
4581     { \@@_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.

```

4582 \int_if_zero:nTF { \l_@@_initial_j_int }
4583 {
4584     \@@_qpoint:n { col - 1 }

```

```

4585     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4586     \dim_sub:Nn \l_@@_x_initial_dim
4587     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4588 }

```

Elsewhere, the brace must be drawn left flush.

```

4589 {
4590     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4591     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4592     \dim_add:Nn \l_@@_x_initial_dim
4593     { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4594 }

```

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

```

4595     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4596     \@@_draw_line:
4597 }

```

```

4598 \cs_new:Npn \@@_colsep:
4599 { \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.

```

4600 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4601 {
4602     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4603     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4604     {
4605         \@@_find_extremities_of_line:nnnn { #1 } { #2 } { 1 } { 1 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4606     \group_begin:
4607     \@@_open_shorten:
4608     \keys_set:nn { nicematrix / xdots } { #3 }
4609     \@@_color:o \l_@@_xdots_color_tl
4610     \@@_actually_draw_Ddots:
4611     \group_end:
4612 }
4613 }

```

The command `\@@_actually_draw_Ddots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

```

4614 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4615 {
4616   \bool_if:NTF \l_@@_initial_open_bool
4617   {
4618     \@@_open_y_initial_dim:
4619     \@@_open_x_initial_dim:
4620   }
4621   { \@@_set_initial_coords_from_anchor:n { south-east } }
4622   \bool_if:NTF \l_@@_final_open_bool
4623   {
4624     \@@_open_x_final_dim:
4625     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4626   }
4627   { \@@_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.

```

4628   \bool_if:NT \l_@@_parallelize_diags_bool
4629   {
4630     \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).

```

4631     \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.

```

4632     {
4633       \dim_gset:Nn \g_@@_delta_x_one_dim
4634       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4635       \dim_gset:Nn \g_@@_delta_y_one_dim
4636       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4637     }

```

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`.

```

4638     {
4639       \dim_compare:nNnF { \g_@@_delta_x_one_dim } = { \c_zero_dim }
4640       {
4641         \dim_set:Nn \l_@@_y_final_dim
4642         {
4643           \l_@@_y_initial_dim +
4644           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4645           \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4646         }
4647       }
4648     }
4649   }
4650   \@@_draw_line:
4651 }

```

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.

```

4652 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4653 {
4654   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4655   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4656   {
4657     \@@_find_extremities_of_line:nnnn { #1 } { #2 } { 1 } { -1 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4658   \group_begin:

```

```

4659         \@@_open_shorten:
4660         \keys_set:nn { nicematrix / xdots } { #3 }
4661         \@@_color:o \l_@@_xdots_color_tl
4662         \@@_actually_draw_Iddots:
4663     \group_end:
4664 }
4665 }

```

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.`

```

4666 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4667 {
4668     \bool_if:NTF \l_@@_initial_open_bool
4669     {
4670         \@@_open_y_initial_dim:
4671         \@@_open_x_initial_dim:
4672     }
4673     { \@@_set_initial_coords_from_anchor:n { south-west } }
4674     \bool_if:NTF \l_@@_final_open_bool
4675     {
4676         \@@_open_y_final_dim:
4677         \@@_open_x_final_dim:
4678     }
4679     { \@@_set_final_coords_from_anchor:n { north-east } }
4680     \bool_if:NT \l_@@_parallelize_diags_bool
4681     {
4682         \int_gincr:N \g_@@_iddots_int
4683         \int_compare:nNnTF { \g_@@_iddots_int } = { \c_one_int }
4684         {
4685             \dim_gset:Nn \g_@@_delta_x_two_dim
4686             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4687             \dim_gset:Nn \g_@@_delta_y_two_dim
4688             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4689         }
4690         {
4691             \dim_compare:nNnF { \g_@@_delta_x_two_dim } = { \c_zero_dim }
4692             {
4693                 \dim_set:Nn \l_@@_y_final_dim
4694                 {
4695                     \l_@@_y_initial_dim +
4696                     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4697                     \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4698                 }
4699             }
4700         }
4701     }
4702     \@@_draw_line:
4703 }

```

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`

```

4704 \cs_new_protected:Npn \@@_draw_line:
4705 {
4706   \pgfrememberpicturepositiononpagetrue
4707   \pgf@relevantforpicturesizefalse
4708   \bool_lazy_or:nnTF
4709     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4710     { \l_@@_dotted_bool }
4711     { \@@_draw_standard_dotted_line: }
4712     { \@@_draw_unstandard_dotted_line: }
4713 }
```

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.

```

4714 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4715 {
4716   \begin { scope }
4717     \@@_draw_unstandard_dotted_line:o
4718     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4719 }
```

We have used the fact that, in PGF, a color name can be put directly in a list of options (that's why we have put directly `\l_@@_xdots_color_tl`).

The argument of `\@@_draw_unstandard_dotted_line:n` is, in fact, the list of options.

```

4720 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4721 {
4722   \@@_draw_unstandard_dotted_line:nooo
4723   { #1 }
4724   \l_@@_xdots_up_tl
4725   \l_@@_xdots_down_tl
4726   \l_@@_xdots_middle_tl
4727 }
4728 \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.

```

4729 \hook_gput_code:nnn { begindocument } { . }
4730 {
4731   \IfPackageLoadedT { tikz }
4732   {
4733     \tikzset
4734     {
4735       @@_node_above / .style = { sloped , above } ,
4736       @@_node_below / .style = { sloped , below } ,
4737       @@_node_middle / .style =
4738       {
```

```

4739         sloped ,
4740         inner~sep = \c_@@_innersep_middle_dim
4741     }
4742 }
4743 }
4744 }

```

```

4745 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4746 {

```

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.

```

4747     \dim_zero_new:N \l_@@_l_dim
4748     \dim_set:Nn \l_@@_l_dim
4749     {
4750         \fp_to_dim:n
4751         {
4752             sqrt
4753             (
4754                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4755                 +
4756                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4757             )
4758         }
4759     }

```

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.

```

4760     \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4761     {
4762         \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4763         \@@_draw_unstandard_dotted_line_i:
4764     }

```

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

```

4765     \bool_if:NT \l_@@_xdots_h_labels_bool
4766     {
4767         \tikzset
4768         {
4769             @@_node_above / .style = { auto = left } ,
4770             @@_node_below / .style = { auto = right } ,
4771             @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4772         }
4773     }
4774     \tl_if_empty:nF { #4 }
4775     { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4776     \draw
4777     [ #1 ]
4778     ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )

```

Be careful: We can’t put `\c_math_toggle_token` instead of `$` in the following lines because we are in the contents of Tikz nodes (and they will be *rescanned* if the Tikz library `babel` is loaded).

```

4779         -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4780         node [ @@_node_below ] { $ \scriptstyle #3 $ }
4781         node [ @@_node_above ] { $ \scriptstyle #2 $ }
4782         ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4783     \end { scope }
4784 }
4785 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }

```

```

4786 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4787 {
4788   \dim_set:Nn \l_tmpa_dim
4789   {
4790     \l_@@_x_initial_dim
4791     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4792     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4793   }
4794   \dim_set:Nn \l_tmpb_dim
4795   {
4796     \l_@@_y_initial_dim
4797     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4798     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4799   }
4800   \dim_set:Nn \l_@@_tmpc_dim
4801   {
4802     \l_@@_x_final_dim
4803     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4804     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4805   }
4806   \dim_set:Nn \l_@@_tmpd_dim
4807   {
4808     \l_@@_y_final_dim
4809     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4810     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4811   }
4812   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4813   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4814   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4815   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4816 }

```

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

```

4817 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4818 {
4819   \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.

```

4820   \dim_zero_new:N \l_@@_l_dim
4821   \dim_set:Nn \l_@@_l_dim
4822   {
4823     \fp_to_dim:n
4824     {
4825       sqrt
4826       (
4827         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4828         +
4829         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4830       )
4831     }
4832   }

```

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.

```

4833   \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4834   {
4835     \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4836     { \@@_draw_standard_dotted_line_i: }
4837   }
4838   \group_end:

```

```

4839 \bool_lazy_all:nF
4840 {
4841   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4842   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4843   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4844 }
4845 { \@@_labels_standard_dotted_line: }
4846 }
4847 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4848 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4849 {

```

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

```

4850 \int_set:Nn \l_tmpa_int
4851 {
4852   \dim_ratio:nn
4853   {
4854     \l_@@_l_dim
4855     - \l_@@_xdots_shorten_start_dim
4856     - \l_@@_xdots_shorten_end_dim
4857   }
4858   { \l_@@_xdots_inter_dim }
4859 }

```

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

```

4860 \dim_set:Nn \l_tmpa_dim
4861 {
4862   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4863   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4864 }
4865 \dim_set:Nn \l_tmpb_dim
4866 {
4867   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4868   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4869 }

```

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.

```

4870 \dim_gadd:Nn \l_@@_x_initial_dim
4871 {
4872   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4873   \dim_ratio:nn
4874   {
4875     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4876     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4877   }
4878   { 2 \l_@@_l_dim }
4879 }
4880 \dim_gadd:Nn \l_@@_y_initial_dim
4881 {
4882   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4883   \dim_ratio:nn
4884   {
4885     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4886     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4887   }
4888   { 2 \l_@@_l_dim }
4889 }
4890 \pgf@relevantforpicturesizefalse
4891 \int_step_inline:nnn { \c_zero_int } { \l_tmpa_int }
4892 {
4893   \pgfpathcircle

```



```

4894         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4895         { \l_@@_xdots_radius_dim }
4896         \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4897         \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4898     }
4899     \pgfusepathqfill
4900 }

4901 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
4902 {
4903     \pgfscope
4904     \pgftransformshift
4905     {
4906         \pgfpointlineattime { 0.5 }
4907         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4908         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4909     }
4910     \fp_set:Nn \l_tmpa_fp
4911     {
4912         atand
4913         (
4914             \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4915             \l_@@_x_final_dim - \l_@@_x_initial_dim
4916         )
4917     }
4918     \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4919     \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4920     \tl_if_empty:NF \l_@@_xdots_middle_tl
4921     {
4922         \begin { pgfscope }
4923         \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4924         \pgfnode
4925         { rectangle }
4926         { center }
4927         {
4928             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4929             {
4930                 \c_math_toggle_token
4931                 \scriptstyle \l_@@_xdots_middle_tl
4932                 \c_math_toggle_token
4933             }
4934         }
4935         { }
4936         {
4937             \pgfsetfillcolor { white }
4938             \pgfusepath { fill }
4939         }
4940         \end { pgfscope }
4941     }
4942     \tl_if_empty:NF \l_@@_xdots_up_tl
4943     {
4944         \pgfnode
4945         { rectangle }
4946         { south }
4947         {
4948             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4949             {
4950                 \c_math_toggle_token
4951                 \scriptstyle \l_@@_xdots_up_tl
4952                 \c_math_toggle_token
4953             }
4954         }
4955         { }

```

```

4956         { \pgfusepath { } }
4957     }
4958     \tl_if_empty:NF \l_@@_xdots_down_tl
4959     {
4960         \pgfnode
4961         { rectangle }
4962         { north }
4963         {
4964             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4965             {
4966                 \c_math_toggle_token
4967                 \scriptstyle \l_@@_xdots_down_tl
4968                 \c_math_toggle_token
4969             }
4970         }
4971         { }
4972         { \pgfusepath { } }
4973     }
4974     \endpgfscope
4975 }

```

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.

```

4976 \hook_gput_code:nnn { begindocument } { . }
4977 {

```

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

```

4978     \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
4979     \cs_new_protected:Npn \@@_Ldots:
4980     { \@@_collect_options:n { \@@_Ldots_i } }
4981     \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4982     {
4983         \int_if_zero:nTF { \c_jCol }
4984         { \@@_error:nn { in~first~col } { \Ldots } }
4985         {
4986             \int_compare:nNnTF { \c_jCol } = { \l_@@_last_col_int }
4987             { \@@_error:nn { in~last~col } { \Ldots } }
4988             {
4989                 \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
4990                 { #1 , down = #2 , up = #3 , middle = #4 }
4991             }
4992         }
4993         \bool_if:NF \l_@@_nullify_dots_bool
4994         { \phantom { \ensuremath { \@@_old_ldots: } } }
4995         \bool_gset_true:N \g_@@_empty_cell_bool
4996     }

4997     \cs_new_protected:Npn \@@_Cdots:

```

```

4998 { \@@_collect_options:n { \@@_Cdots_i } }
4999 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5000 {
5001   \int_if_zero:nTF { \c@jCol }
5002   { \@@_error:nn { in~first~col } { \Cdots } }
5003   {
5004     \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5005     { \@@_error:nn { in~last~col } { \Cdots } }
5006     {
5007       \@@_instruction_of_type:nnn { \c_false_bool } { Cdots }
5008       { #1 , down = #2 , up = #3 , middle = #4 }
5009     }
5010   }
5011   \bool_if:NF \l_@@_nullify_dots_bool
5012   { \phantom { \ensuremath { \@@_old_cdots: } } }
5013   \bool_gset_true:N \g_@@_empty_cell_bool
5014 }

5015 \cs_new_protected:Npn \@@_Vdots:
5016 { \@@_collect_options:n { \@@_Vdots_i } }
5017 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5018 {
5019   \int_if_zero:nTF { \c@iRow }
5020   { \@@_error:nn { in~first~row } { \Vdots } }
5021   {
5022     \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
5023     { \@@_error:nn { in~last~row } { \Vdots } }
5024     {
5025       \@@_instruction_of_type:nnn { \c_false_bool } { Vdots }
5026       { #1 , down = #2 , up = #3 , middle = #4 }
5027     }
5028   }
5029   \bool_if:NF \l_@@_nullify_dots_bool
5030   { \phantom { \ensuremath { \@@_old_vdots: } } }
5031   \bool_gset_true:N \g_@@_empty_cell_bool
5032 }

5033 \cs_new_protected:Npn \@@_Ddots:
5034 { \@@_collect_options:n { \@@_Ddots_i } }
5035 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5036 {
5037   \int_case:nnF \c@iRow
5038   {
5039     0 { \@@_error:nn { in~first~row } { \Ddots } }
5040     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5041   }
5042   {
5043     \int_case:nnF \c@jCol
5044     {
5045       0 { \@@_error:nn { in~first~col } { \Ddots } }
5046       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5047     }
5048     {
5049       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5050       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5051       { #1 , down = #2 , up = #3 , middle = #4 }
5052     }
5053   }
5054 }
5055 \bool_if:NF \l_@@_nullify_dots_bool
5056 { \phantom { \ensuremath { \@@_old_ddots: } } }
5057 \bool_gset_true:N \g_@@_empty_cell_bool
5058 }

```

```

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

```

End of the \AddToHook.

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

```

5085 \keys_define:nn { nicematrix / Ddots }
5086 {
5087   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5088   draw-first .default:n = true ,
5089   draw-first .value_forbidden:n = true
5090 }

```

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

```

5091 \cs_new_protected:Npn \@@_Hspace:
5092 {
5093   \bool_gset_true:N \g_@@_empty_cell_bool
5094   \hspace
5095 }

```

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.

```

5096 \cs_set_eq:NN \@@_old_multicolumn: \multicolumn

```

The command \@@_Hdotsfor will be linked to \Hdotsfor in {NiceArrayWithDelims}. Tikz nodes are created also in the implicit cells of the \Hdotsfor (maybe we should modify that point).

This command must *not* be protected since it begins with \multicolumn.

```

5097 \cs_new:Npn \@@_Hdotsfor:
5098 {
5099   \bool_lazy_and:nnTF
5100   { \int_if_zero_p:n { \c@jCol } }
5101   { \int_if_zero_p:n { \l_@@_first_col_int } }
5102   {
5103     \bool_if:NTF \g_@@_after_col_zero_bool
5104     {
5105       \multicolumn { 1 } { c } { }
5106       \@@_Hdotsfor_i:

```

```

5107     }
5108     { \@@_fatal:n { Hdotsfor~in~col-0 } }
5109 }
5110 {
5111     \multicolumn { 1 } { c } { }
5112     \@@_Hdotsfor_i:
5113 }
5114 }

```

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:`).

```

5115 \hook_gput_code:nnn { begindocument } { . }
5116 {

```

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

```

5117     \cs_new_protected:Npn \@@_Hdotsfor_i:
5118     { \@@_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`).

```

5119     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5120     \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5121     {
5122         \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5123         {
5124             \@@_Hdotsfor:nnnn
5125             { \int_use:N \c@iRow }
5126             { \int_use:N \c@jCol }
5127             { #2 }
5128             {
5129                 #1 , #3 ,
5130                 down = \exp_not:n { #4 } ,
5131                 up = \exp_not:n { #5 } ,
5132                 middle = \exp_not:n { #6 }
5133             }
5134         }
5135         \prg_replicate:nn { #2 - 1 }
5136         {
5137             &
5138             \multicolumn { 1 } { c } { }
5139             \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5140         }
5141     }
5142 }

```

```

5143 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5144 {
5145     \bool_set_false:N \l_@@_initial_open_bool
5146     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5147     \int_set:Nn \l_@@_initial_i_int { #1 }
5148     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5149     \int_compare:nNnTF { #2 } = { \c_one_int }
5150     {
5151         \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5152         \bool_set_true:N \l_@@_initial_open_bool
5153     }
5154     {
5155         \cs_if_exist:cTF
5156         {

```

```

5157         pgf @ sh @ ns @ \@@_env:
5158         - \int_use:N \l_@@_initial_i_int
5159         - \int_eval:n { #2 - 1 }
5160     }
5161     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5162     {
5163         \int_set:Nn \l_@@_initial_j_int { #2 }
5164         \bool_set_true:N \l_@@_initial_open_bool
5165     }
5166 }
5167 \int_compare:nNnTF { #2 + #3 - 1 } = { \c@jCol }
5168 {
5169     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5170     \bool_set_true:N \l_@@_final_open_bool
5171 }
5172 {
5173     \cs_if_exist:cTF
5174     {
5175         pgf @ sh @ ns @ \@@_env:
5176         - \int_use:N \l_@@_final_i_int
5177         - \int_eval:n { #2 + #3 }
5178     }
5179     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5180     {
5181         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5182         \bool_set_true:N \l_@@_final_open_bool
5183     }
5184 }
5185 \group_begin:
5186 \@@_open_shorten:
5187 \int_if_zero:nTF { #1 }
5188 { \color { nicematrix-first-row } }
5189 {
5190     \int_compare:nNnT { #1 } = { \g_@@_row_total_int }
5191     { \color { nicematrix-last-row } }
5192 }
5193 \keys_set:nn { nicematrix / xdots } { #4 }
5194 \@@_color:o \l_@@_xdots_color_tl
5195 \@@_actually_draw_Ldots:
5196 \group_end:

```

We declare all the cells concerned by the `\Hdotsfor` as “dotted” (for the dotted lines created by `\Cdots`, `\Ldots`, etc., this job is done by `\@@_find_extremities_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5197     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5198     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5199 }

```

```

5200 \hook_gput_code:nnn { begindocument } { . }
5201 {
5202     \cs_new_protected:Npn \@@_Vdotsfor:
5203     { \@@_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`).

```

5204     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5205     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5206     {
5207         \bool_gset_true:N \g_@@_empty_cell_bool
5208         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5209         {
5210             \@@_Vdotsfor:nnnn

```

```

5211         { \int_use:N \c@iRow }
5212         { \int_use:N \c@jCol }
5213         { #2 }
5214         {
5215             #1 , #3 ,
5216             down = \exp_not:n { #4 } ,
5217             up = \exp_not:n { #5 } ,
5218             middle = \exp_not:n { #6 }
5219         }
5220     }
5221 }
5222 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5223 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5224 {
5225     \bool_set_false:N \l_@@_initial_open_bool
5226     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5227     \int_set:Nn \l_@@_initial_j_int { #2 }
5228     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5229     \int_compare:nNnTF { #1 } = { \c_one_int }
5230     {
5231         \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5232         \bool_set_true:N \l_@@_initial_open_bool
5233     }
5234     {
5235         \cs_if_exist:cTF
5236         {
5237             pgf @ sh @ ns @ \@@_env:
5238             - \int_eval:n { #1 - 1 }
5239             - \int_use:N \l_@@_initial_j_int
5240         }
5241         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5242         {
5243             \int_set:Nn \l_@@_initial_i_int { #1 }
5244             \bool_set_true:N \l_@@_initial_open_bool
5245         }
5246     }
5247     \int_compare:nNnTF { #1 + #3 - 1 } = { \c@iRow }
5248     {
5249         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5250         \bool_set_true:N \l_@@_final_open_bool
5251     }
5252     {
5253         \cs_if_exist:cTF
5254         {
5255             pgf @ sh @ ns @ \@@_env:
5256             - \int_eval:n { #1 + #3 }
5257             - \int_use:N \l_@@_final_j_int
5258         }
5259         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5260         {
5261             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5262             \bool_set_true:N \l_@@_final_open_bool
5263         }
5264     }

```

```

5265 \group_begin:
5266 \@@_open_shorten:
5267 \int_if_zero:nTF { #2 }
5268   { \color { nicematrix-first-col } }
5269   {
5270     \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
5271     { \color { nicematrix-last-col } }
5272   }
5273 \keys_set:nn { nicematrix / xdots } { #4 }
5274 \@@_color:o \l_@@_xdots_color_tl
5275 \bool_if:NTF \l_@@_Vbrace_bool
5276   { \@@_actually_draw_Vbrace: }
5277   { \@@_actually_draw_Vdots: }
5278 \group_end:

```

We declare all the cells concerned by the `\Vdotsfor` as “dotted” (for the dotted lines created by `\Cdots`, `\Ldots`, etc., this job is done by `\@@_find_extremities_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5279 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5280   { \cs_set_nopar:cpn { @@_dotted_ ##1 - #2 } { } }
5281 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5282 \NewDocumentCommand \@@_rotate: { 0 { } }
5283   {
5284     \bool_gset_true:N \g_@@_rotate_bool
5285     \keys_set:nn { nicematrix / rotate } { #1 }
5286     \ignorespaces
5287   }

5288 \keys_define:nn { nicematrix / rotate }
5289   {
5290     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5291     c .value_forbidden:n = true ,
5292     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5293   }

```

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

```

5294 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop

```

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


```

5295 {
5296   \tl_if_empty:nTF { #2 }
5297     { #1 }
5298     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5299 }
5300 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5301 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

With the following construction, the command `\@@_double_int_eval:n` is applied to both arguments before the application of `\@@_line_i:nn` (the construction uses the fact the `\@@_line_i:nn` is protected and that `\@@_double_int_eval:n` is fully expandable).

```

5302 \hook_gput_code:nnn { begindocument } { . }
5303 {

```

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

```

5304   \tl_set_rescan:Nnn \l_tmpa_tl { }
5305   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5306   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5307     {
5308       \group_begin:
5309       \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5310       \@@_color:o \l_@@_xdots_color_tl
5311       \use:e
5312       {
5313         \@@_line_i:nn
5314         { \@@_double_int_eval:n #2 - \q_stop }
5315         { \@@_double_int_eval:n #3 - \q_stop }
5316       }
5317       \group_end:
5318     }
5319 }
5320 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5321 {
5322   \bool_set_false:N \l_@@_initial_open_bool
5323   \bool_set_false:N \l_@@_final_open_bool
5324   \bool_lazy_or:nnTF
5325     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5326     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5327     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

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

```

5328   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5329 }
5330 \hook_gput_code:nnn { begindocument } { . }
5331 {
5332   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5333   {

```

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:`.

```

5334     \c_@@_pgfortikzpicture_tl
5335     \@@_draw_line_iii:nn { #1 } { #2 }
5336     \c_@@_endpgfortikzpicture_tl
5337   }
5338 }

```

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

```

5339 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5340 {
5341   \pgfrememberpicturepositiononpagetrue

```

```

5342 \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5343 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5344 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5345 \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5346 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5347 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5348 \@@_draw_line:
5349 }

```

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.

```

5350 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNnT { \c@iRow } < }
5351 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNnF { \c@jCol } < }

```

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

```

5352 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5353 {
5354   \tl_gput_right:Ne \g_@@_row_style_tl
5355   {

```

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

```

5356 \exp_not:N
5357 \@@_if_row_less_than:nn
5358 { \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`).

```

5359 {
5360   \exp_not:N
5361   \@@_if_col_greater_than:nn
5362   { \int_eval:n { \c@jCol } }
5363   { \exp_not:n { #1 } \scan_stop: }
5364 }
5365 }
5366 }
5367 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

```

```

5368 \keys_define:nn { nicematrix / RowStyle }
5369 {
5370   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5371   cell-space-top-limit .value_required:n = true ,
5372   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5373   cell-space-bottom-limit .value_required:n = true ,

```

```

5374 cell-space-limits .meta:n =
5375 {
5376     cell-space-top-limit = #1 ,
5377     cell-space-bottom-limit = #1 ,
5378 } ,
5379 color .tl_set:N = \l_@@_color_tl ,
5380 color .value_required:n = true ,
5381 bold .bool_set:N = \l_@@_bold_row_style_bool ,
5382 bold .default:n = true ,
5383 nb-rows .code:n =
5384     \str_if_eq:eeTF { #1 } { * }
5385     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5386     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5387 nb-rows .value_required:n = true ,
5388 fill .tl_set:N = \l_@@_fill_tl ,
5389 fill .value_required:n = true ,
5390 opacity .tl_set:N = \l_@@_opacity_tl ,
5391 opacity .value_required:n = true ,
5392 rowcolor .tl_set:N = \l_@@_fill_tl ,
5393 rowcolor .value_required:n = true ,
5394 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5395 rounded-corners .default:n = 4 pt ,
5396 unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5397 }

```

```

5398 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5399 {
5400     \group_begin:
5401     \tl_clear:N \l_@@_fill_tl
5402     \tl_clear:N \l_@@_opacity_tl
5403     \tl_clear:N \l_@@_color_tl
5404     \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5405     \dim_zero:N \l_@@_rounded_corners_dim
5406     \dim_zero:N \l_tmpa_dim
5407     \dim_zero:N \l_tmpb_dim
5408     \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5409     \tl_if_empty:NF \l_@@_fill_tl
5410     {
5411         \@@_add_opacity_to_fill:
5412         \tl_gput_right:Ne \g_@@_pre_code_before_tl
5413         {

```

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

```

5414         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5415         { \int_use:N \c_iRow - \int_use:N \c_jCol }
5416         {
5417             \int_eval:n { \c_iRow + \l_@@_key_nb_rows_int - 1 }
5418             - *
5419         }
5420         { \dim_use:N \l_@@_rounded_corners_dim }
5421     }
5422 }
5423 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5424     \dim_compare:nNnT { \l_tmpa_dim } > { \c_zero_dim }
5425     {
5426         \@@_put_in_row_style:e
5427         {
5428             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5429             {

```

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

```

5430         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5431         { \dim_use:N \l_tmpa_dim }
5432     }
5433 }
5434 }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

5435     \dim_compare:nNnT { \l_tmpb_dim } > { \c_zero_dim }
5436     {
5437         \@@_put_in_row_style:e
5438         {
5439             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5440             {
5441                 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5442                 { \dim_use:N \l_tmpb_dim }
5443             }
5444         }
5445     }

```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

5446     \tl_if_empty:NF \l_@@_color_tl
5447     {
5448         \@@_put_in_row_style:e
5449         {
5450             \mode_leave_vertical:
5451             \@@_color:n { \l_@@_color_tl }
5452         }
5453     }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5454     \bool_if:NT \l_@@_bold_row_style_bool
5455     {
5456         \@@_put_in_row_style:n
5457         {
5458             \exp_not:n
5459             {
5460                 \if_mode_math:
5461                 \c_math_toggle_token
5462                 \bfseries \boldmath
5463                 \c_math_toggle_token
5464             \else:
5465                 \bfseries \boldmath
5466             \fi:
5467         }
5468     }
5469 }
5470 \group_end:
5471 \g_@@_row_style_tl
5472 \ignorespaces
5473 }

```

The following commande must *not* be protected.

```

5474 \cs_new:Npn \@@_rounded_from_row:n #1
5475 {
5476     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

In the following code, the “`- 1`” is *not* a subtraction.

```

5477     { \int_eval:n { #1 } - 1 }
5478     {
5479         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5480         - \exp_not:n { \int_use:N \c@jCol }
5481     }
5482     { \dim_use:N \l_@@_rounded_corners_dim }
5483 }

```

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).

```
5484 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5485 {
```

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.

```
5486 \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`.

```
5487 \str_if_in:nnF { #1 } { !! }
5488 {
5489 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```
5490 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5491 }
5492 \int_if_zero:nTF { \l_tmpa_int }
```

First, the case where the color is a *new* color (not in the sequence).

```
5493 {
5494 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5495 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5496 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5497 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5498 }
5499 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5500 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

The following command must be used within a `\pgfpicture`.

```
5501 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5502 {
5503 \dim_compare:nNnT { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim }
5504 {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5505     \group_begin:
5506     \pgfsetcornersarced
5507     {
5508         \pgfpoint
5509         { \l_@@_tab_rounded_corners_dim }
5510         { \l_@@_tab_rounded_corners_dim }
5511     }

```

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.

```

5512     \bool_if:NTF \l_@@_hvlines_bool
5513     {
5514         \pgfpathrectanglecorners
5515         {
5516             \pgfpointadd
5517             { \@@_qpoint:n { row-1 } }
5518             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5519         }
5520         {
5521             \pgfpointadd
5522             {
5523                 \@@_qpoint:n
5524                 { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5525             }
5526             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5527         }
5528     }
5529     {
5530         \pgfpathrectanglecorners
5531         { \@@_qpoint:n { row-1 } }
5532         {
5533             \pgfpointadd
5534             {
5535                 \@@_qpoint:n
5536                 { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5537             }
5538             { \pgfpoint \c_zero_dim \arrayrulewidth }
5539         }
5540     }
5541     \pgfusepath { clip }
5542     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5543     }
5544 }

```

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`).

```

5545 \cs_new_protected:Npn \@@_actually_color:
5546 {
5547     \pgfpicture
5548     \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.

```

5549     \@@_clip_with_rounded_corners:
5550     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5551     {
5552         \int_compare:nNnTF { ##1 } = { \c_one_int }

```

```

5553     {
5554         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5555         \use:c { g_@@_color _ 1 _tl }
5556         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5557     }
5558     {
5559         \begin { pgfscope }
5560             \@@_color_opacity: ##2
5561             \use:c { g_@@_color _ ##1 _tl }
5562             \tl_gclear:c { g_@@_color _ ##1 _tl }
5563             \pgfusepath { fill }
5564         \end { pgfscope }
5565     }
5566 }
5567 \endpgfpicture
5568 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5569 \cs_new_protected:Npn \@@_color_opacity:
5570 {
5571     \peek_meaning:NTF [
5572         { \@@_color_opacity:w }
5573         { \@@_color_opacity:w [ ] }
5574     }

```

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 currying.

```

5575 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5576 {
5577     \tl_clear:N \l_tmpa_tl
5578     \keys_set_known:nnN { nicematrix / color-opacity } { #1 } \l_tmpb_tl

```

`\l_tmpa_tl` (if not empty) is now the opacity and `\l_tmpb_tl` (if not empty) is now the colorimetric space.

```

5579     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5580     \tl_if_empty:NTF \l_tmpb_tl
5581         { \@declaredcolor }
5582         { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5583 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5584 \keys_define:nn { nicematrix / color-opacity }
5585 {
5586     opacity .tl_set:N          = \l_tmpa_tl ,
5587     opacity .value_required:n = true
5588 }

```

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

```

5589 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5590 {
5591     \def \l_@@_rows_tl { #1 }
5592     \def \l_@@_cols_tl { #2 }
5593     \@@_cartesian_path:
5594 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5595 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5596 {
5597     \tl_if_blank:nF { #2 }

```

```

5598 {
5599   \@@_add_to_colors_seq:en
5600   { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5601   { \@@_cartesian_color:nn { #3 } { - } }
5602 }
5603 }

```

Here an example: `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5604 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5605 {
5606   \tl_if_blank:nF { #2 }
5607   {
5608     \@@_add_to_colors_seq:en
5609     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5610     { \@@_cartesian_color:nn { - } { #3 } }
5611   }
5612 }

```

Here is an example: `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5613 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5614 {
5615   \tl_if_blank:nF { #2 }
5616   {
5617     \@@_add_to_colors_seq:en
5618     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5619     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5620   }
5621 }

```

The last argument is the radius of the corners of the rectangle.

```

5622 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5623 {
5624   \tl_if_blank:nF { #2 }
5625   {
5626     \@@_add_to_colors_seq:en
5627     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5628     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5629   }
5630 }

```

The last argument is the radius of the corners of the rectangle.

```

5631 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5632 {
5633   \@@_cut_on_hyphen:w #1 \q_stop
5634   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5635   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5636   \@@_cut_on_hyphen:w #2 \q_stop
5637   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5638   \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`.

```

5639   \@@_cartesian_path:n { #3 }
5640 }

```

Here is an example: `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5641 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5642 {
5643   \clist_map_inline:nn { #3 }
5644   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5645 }

```



```

5646 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5647 {
5648   \int_step_inline:nn { \c@iRow }
5649   {
5650     \int_step_inline:nn { \c@jCol }
5651     {
5652       \int_if_even:nTF { #####1 + ##1 }
5653       { \@@_cellcolor [ #1 ] { #2 } }
5654       { \@@_cellcolor [ #1 ] { #3 } }
5655       { ##1 - #####1 }
5656     }
5657   }
5658 }

```

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”.

```

5659 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5660 {
5661   \@@_rectanglecolor [ #1 ] { #2 }
5662   { 1 - 1 }
5663   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5664 }

```

```

5665 \keys_define:nn { nicematrix / rowcolors }
5666 {
5667   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5668   respect-blocks .default:n = true ,
5669   cols .tl_set:N = \l_@@_cols_tl ,
5670   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5671   restart .default:n = true ,
5672   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5673 }

```

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*.

```

5674 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5675 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5676   \group_begin:
5677   \seq_clear_new:N \l_@@_colors_seq
5678   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5679   \tl_clear_new:N \l_@@_cols_tl
5680   \tl_set:Nn \l_@@_cols_tl { - }
5681   \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).

```

5682   \int_zero_new:N \l_@@_color_int
5683   \int_set_eq:NN \l_@@_color_int \c_one_int
5684   \bool_if:NT \l_@@_respect_blocks_bool
5685   {

```

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`).

```

5686     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5687     \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5688     { \@@_not_in_exterior_p:nnnnn ##1 }
5689   }
5690   \pgfpicture
5691   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5692   \clist_map_inline:nn { #2 }
5693   {
5694     \tl_set:Nn \l_tmpa_tl { ##1 }
5695     \tl_if_in:NnTF \l_tmpa_tl { - }
5696     { \@@_cut_on_hyphen:w ##1 \q_stop }
5697     { \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.

```

5698     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5699     \int_set:Nn \l_@@_color_int
5700     { \bool_if:NNTF \l_@@_rowcolors_restart_bool { 1 } { \l_tmpa_tl } }
5701     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5702     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5703     {

```

We will compute in `\l_tmpb_int` the last row of the "block".

```

5704         \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).

```

5705         \bool_if:NT \l_@@_respect_blocks_bool
5706         {
5707           \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5708           { \@@_intersect_our_row_p:nnnnn #####1 }
5709           \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5710         }
5711         \tl_set:Nn \l_@@_rows_tl
5712         { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5713         \tl_set:Nn \l_@@_color_tl
5714         {
5715           \@@_color_index:n
5716           {
5717             \int_mod:nn
5718             { \l_@@_color_int - 1 }
5719             { \seq_count:N \l_@@_colors_seq }
5720             + 1
5721           }
5722         }
5723         \tl_if_empty:NF \l_@@_color_tl
5724         {
5725           \@@_add_to_colors_seq:ee
5726           { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5727           { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5728         }
5729         \int_incr:N \l_@@_color_int
5730         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5731       }
5732     }
5733   \endpgfpicture
5734   \group_end:
5735 }

```

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.

```
5736 \cs_new:Npn \@@_color_index:n #1
5737 {
```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```
5738 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5739 { \@@_color_index:n { #1 - 1 } }
5740 { \seq_item:Nn \l_@@_colors_seq { #1 } }
5741 }
```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by currying.

```
5742 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5743 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }
```

The braces around `#3` and `#4` are mandatory.

```
5744 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5745 {
5746   \int_compare:nNtT { #3 } > { \l_tmpb_int }
5747   { \int_set:Nn \l_tmpb_int { #3 } }
5748 }
```

```
5749 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5750 {
5751   \int_if_zero:nTF { #4 }
5752   { \prg_return_false: }
5753   {
5754     \int_compare:nNtTF { #2 } > { \c@jCol }
5755     { \prg_return_false: }
5756     { \prg_return_true: }
5757   }
5758 }
```

The following command return true when the block intersects the row `\l_tmpa_int`.

```
5759 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5760 {
5761   \int_compare:nNtTF { #1 } > { \l_tmpa_int }
5762   { \prg_return_false: }
5763   {
5764     \int_compare:nNtTF { \l_tmpa_int } > { #3 }
5765     { \prg_return_false: }
5766     { \prg_return_true: }
5767   }
5768 }
```

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`).

```
5769 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5770 {
5771   \dim_compare:nNtTF { #1 } = { \c_zero_dim }
5772   {
5773     \bool_if:NtF \l_@@_nocolor_used_bool
5774     { \@@_cartesian_path_normal_ii: }
```

```

5775     {
5776         \clist_if_empty:NTF \l_@@_corners_cells_clist
5777         { \@@_cartesian_path_normal_i:n { #1 } }
5778         { \@@_cartesian_path_normal_ii: }
5779     }
5780 }
5781 { \@@_cartesian_path_normal_i:n { #1 } }
5782 }

```

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.

```

5783 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5784 {
5785     \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5786     \clist_map_inline:Nn \l_@@_cols_tl
5787     {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

5788         \def \l_tmpa_tl { ##1 }
5789         \tl_if_in:NnTF \l_tmpa_tl { - }
5790         { \@@_cut_on_hyphen:w ##1 \q_stop }
5791         { \def \l_tmpb_tl { ##1 } } % 2025-04-16
5792         \tl_if_empty:NTF \l_tmpa_tl
5793         { \def \l_tmpa_tl { 1 } }
5794         {
5795             \str_if_eq:eeT \l_tmpa_tl { * }
5796             { \def \l_tmpa_tl { 1 } }
5797         }
5798         \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_col_total_int }
5799         { \@@_error:n { Invalid~col~number } }
5800         \tl_if_empty:NTF \l_tmpb_tl
5801         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5802         {
5803             \str_if_eq:eeT \l_tmpb_tl { * }
5804             { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5805         }
5806         \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_col_total_int }
5807         { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5808         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5809         \@@_qpoint:n { col - \l_tmpa_tl }
5810         \int_compare:nNnTF { \l_@@_first_col_int } = { \l_tmpa_tl }
5811         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5812         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5813         \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5814         \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.

```

5815     \clist_map_inline:Nn \l_@@_rows_tl
5816     {
5817         \def \l_tmpa_tl { #####1 }
5818         \tl_if_in:NnTF \l_tmpa_tl { - }
5819         { \@@_cut_on_hyphen:w #####1 \q_stop }
5820         { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5821         \tl_if_empty:NTF \l_tmpa_tl
5822         { \def \l_tmpa_tl { 1 } }
5823         {
5824             \str_if_eq:eeT \l_tmpa_tl { * }
5825             { \def \l_tmpa_tl { 1 } }
5826         }
5827         \tl_if_empty:NTF \l_tmpb_tl

```

```

5828         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5829         {
5830             \str_if_eq:eeT \l_tmpb_tl { * }
5831             { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5832         }
5833         \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_row_total_int }
5834         { \@@_error:n { Invalid-row-number } }
5835         \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_row_total_int }
5836         { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_row_total_int } }

```

Now, the numbers of both rows are in `\l_tmpa_tl` and `\l_tmpb_tl`.

```

5837         \cs_if_exist:cF
5838         { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5839         {
5840             \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5841             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5842             \@@_qpoint:n { row - \l_tmpa_tl }
5843             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5844             \pgfpathrectanglecorners
5845             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5846             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5847         }
5848     }
5849 }
5850 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5851 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5852 {
5853     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5854     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5855     \clist_map_inline:Nn \l_@@_cols_tl
5856     {
5857         \@@_qpoint:n { col - ##1 }
5858         \int_compare:nNnTF { \l_@@_first_col_int } = { ##1 }
5859         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5860         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5861         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5862         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5863     \clist_map_inline:Nn \l_@@_rows_tl
5864     {
5865         \@@_if_in_corner:nF { #####1 - ##1 }
5866         {
5867             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5868             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5869             \@@_qpoint:n { row - #####1 }
5870             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5871             \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5872             {
5873                 \pgfpathrectanglecorners
5874                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5875                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5876             }
5877         }
5878     }
5879 }
5880 }

```

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`).

```
5881 \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

```
5882 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5883 {
5884   \bool_set_true:N \l_@@_nocolor_used_bool
5885   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5886   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
```

We begin the loop over the columns.

```
5887   \clist_map_inline:Nn \l_@@_rows_tl
5888   {
5889     \clist_map_inline:Nn \l_@@_cols_tl
5890     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - ####1 } { } }
5891   }
5892 }
```

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`.

```
5893 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5894 {
5895   \clist_set_eq:NN \l_tmpa_clist #1
5896   \clist_clear:N #1
5897   \clist_map_inline:Nn \l_tmpa_clist
5898   {
```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```
5899     \def \l_tmpa_tl { ##1 }
5900     \tl_if_in:NnTF \l_tmpa_tl { - }
5901     { \@@_cut_on_hyphen:w ##1 \q_stop }
5902     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5903     \bool_lazy_or:nnT
5904     { \str_if_eq_p:ee \l_tmpa_tl { * } }
5905     { \tl_if_blank_p:o \l_tmpa_tl }
5906     { \def \l_tmpa_tl { 1 } }
5907     \bool_lazy_or:nnT
5908     { \str_if_eq_p:ee \l_tmpb_tl { * } }
5909     { \tl_if_blank_p:o \l_tmpb_tl }
5910     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5911     \int_compare:nNnT { \l_tmpb_tl } > { #2 }
5912     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5913     \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
5914     { \clist_put_right:Nn #1 { ####1 } }
5915   }
5916 }
```

The following command will be linked to `\cellcolor` in the tabular.

```
5917 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5918 {
5919   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5920   {
```

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`).

```
5921     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5922     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5923   }
5924   \ignorespaces
5925 }
```

The following command will be linked to `\rowcolor` in the tabular.

```

5926 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5927 {
5928   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5929   {
5930     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5931     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5932     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5933   }
5934   \ignorespaces
5935 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5936 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5937 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around #2 and #3 are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

5938 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5939 {

```

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`.

```

5940   \seq_gclear:N \g_tmpa_seq
5941   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5942     { \@@_rowlistcolors_tabular:nnnn #1 }
5943   \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`.

```

5944   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5945   {
5946     { \int_use:N \c@iRow }
5947     { \exp_not:n { #1 } }
5948     { \exp_not:n { #2 } }
5949     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5950   }
5951   \ignorespaces
5952 }

```

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`).

```

5953 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
5954 {
5955   \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).

```

5956   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5957   {
5958     \tl_gput_right:Ne \g_@@_pre_code_before_tl

```

```

5959     {
5960         \@_rowlistcolors
5961         [ \exp_not:n { #2 } ]
5962         { #1 - \int_eval:n { \c@iRow - 1 } }
5963         { \exp_not:n { #3 } }
5964         [ \exp_not:n { #4 } ]
5965     }
5966 }
5967 }

```

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.

```

5968 \cs_new_protected:Npn \@_clear_rowlistcolors_seq:
5969 {
5970     \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5971     { \@_rowlistcolors_tabular_ii:nnnn ##1 }
5972     \seq_gclear:N \g_@@_rowlistcolors_seq
5973 }
5974 \cs_new_protected:Npn \@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5975 {
5976     \tl_gput_right:Nn \g_@@_pre_code_before_tl
5977     { \@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5978 }

```

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.

```

5979 \NewDocumentCommand \@_columncolor_preamble { 0 { } m }
5980 {

```

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).

```

5981     \int_compare:nNtT { \c@jCol } > { \g_@@_col_total_int }
5982     {

```

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).

```

5983     \tl_gput_left:Ne \g_@@_pre_code_before_tl
5984     {
5985         \exp_not:N \columncolor [ #1 ]
5986         { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5987     }
5988 }
5989 }
5990 \cs_new_protected:Npn \@_EmptyColumn:n #1
5991 {
5992     \clist_map_inline:nn { #1 }
5993     {
5994         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
5995         { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
5996         \columncolor { nocolor } { ##1 }
5997     }
5998 }

```



```

5999 \cs_new_protected:Npn \@@_EmptyRow:n #1
6000 {
6001   \clist_map_inline:nn { #1 }
6002   {
6003     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6004     { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6005     \rowcolor { nocolor } { ##1 }
6006   }
6007 }

```

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`.

```

6008 \cs_set_eq:NN \OnlyMainNiceMatrix \use:n

```

Another definition of `\OnlyMainNiceMatrix` will be linked to the command in the environments of `nicematrix`. Here is that definition, called `\@@_OnlyMainNiceMatrix:n`.

```

6009 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6010 {
6011   \int_if_zero:nTF { \l_@@_first_col_int }
6012   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6013   {
6014     \int_if_zero:nTF { \c_jCol }
6015     {
6016       \int_compare:nNnF { \c_iRow } = { -1 }
6017       {
6018         \int_compare:nNnF { \c_iRow } = { \l_@@_last_row_int - 1 }
6019         { #1 }
6020       }
6021     }
6022     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6023   }
6024 }

```

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.

```

6025 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6026 {
6027   \int_if_zero:nF { \c_iRow }
6028   {
6029     \int_compare:nNnF { \c_iRow } = { \l_@@_last_row_int }
6030     {
6031       \int_compare:nNnT { \c_jCol } > { \c_zero_int }
6032       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6033     }
6034   }
6035 }

```

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`.

```

6036 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6037 {
6038   \IfPackageLoadedTF { tikz }
6039   {
6040     \IfPackageLoadedTF { booktabs }
6041     { #2 }
6042     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6043   }
6044   { \@@_error:nn { TopRule~without~tikz } { #1 } }
6045 }

6046 \NewExpandableDocumentCommand { \@@_TopRule } { } { }
6047 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }

6048 \cs_new:Npn \@@_TopRule_i:
6049 {
6050   \noalign \bgroup
6051   \peek_meaning:NTF [
6052   { \@@_TopRule_ii: }
6053   { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6054 }

6055 \NewDocumentCommand \@@_TopRule_ii: { o }
6056 {
6057   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6058   {
6059     \@@_hline:n
6060     {
6061       position = \int_eval:n { \c@iRow + 1 } ,
6062       tikz =
6063       {
6064         line-width = #1 ,
6065         yshift = 0.25 \arrayrulewidth ,
6066         shorten-< = - 0.5 \arrayrulewidth
6067       } ,
6068       total-width = #1
6069     }
6070   }
6071   \skip_vertical:n { \belowrulesep + #1 }
6072   \egroup
6073 }

6074 \NewExpandableDocumentCommand { \@@_BottomRule } { } { }
6075 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }

6076 \cs_new:Npn \@@_BottomRule_i:
6077 {
6078   \noalign \bgroup
6079   \peek_meaning:NTF [
6080   { \@@_BottomRule_ii: }
6081   { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6082 }

6083 \NewDocumentCommand \@@_BottomRule_ii: { o }
6084 {
6085   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6086   {
6087     \@@_hline:n
6088     {
6089       position = \int_eval:n { \c@iRow + 1 } ,
6090       tikz =
6091       {
6092         line-width = #1 ,

```

```

6093         yshift = 0.25 \arrayrulewidth ,
6094         shorten~< = - 0.5 \arrayrulewidth
6095     } ,
6096     total-width = #1 ,
6097 }
6098 }
6099 \skip_vertical:N \aboverulesep
6100 \@@_create_row_node_i:
6101 \skip_vertical:n { #1 }
6102 \egroup
6103 }
6104 \NewExpandableDocumentCommand { \@@_MidRule } { }
6105 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6106 \cs_new:Npn \@@_MidRule_i:
6107 {
6108     \noalign \bgroup
6109     \peek_meaning:NTF [
6110         { \@@_MidRule_ii: }
6111         { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6112     ]
6113 \NewDocumentCommand \@@_MidRule_ii: { o }
6114 {
6115     \skip_vertical:N \aboverulesep
6116     \@@_create_row_node_i:
6117     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6118     {
6119         \@@_hline:n
6120         {
6121             position = \int_eval:n { \c@iRow + 1 } ,
6122             tikz =
6123             {
6124                 line-width = #1 ,
6125                 yshift = 0.25 \arrayrulewidth ,
6126                 shorten~< = - 0.5 \arrayrulewidth
6127             } ,
6128             total-width = #1 ,
6129         }
6130     }
6131     \skip_vertical:n { \belowrulesep + #1 }
6132     \egroup
6133 }

```

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.

```

6134 \keys_define:nn { nicematrix / Rules }
6135 {
6136     position .int_set:N = \l_@@_position_int ,
6137     position .value_required:n = true ,
6138     start .int_set:N = \l_@@_start_int ,
6139     end .code:n =
6140     \bool_lazy_or:nnTF
6141     { \tl_if_empty_p:n { #1 } }
6142     { \str_if_eq_p:ee { #1 } { last } }
6143     { \int_set_eq:NN \l_@@_end_int \c@jCol }
6144     { \int_set:Nn \l_@@_end_int { #1 } }
6145 }

```

It's possible that the rule won't be drawn continuously from `start` to `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That's why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```

6146 \keys_define:nn { nicematrix / RulesBis }
6147   {
6148     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6149     multiplicity .initial:n = 1 ,
6150     dotted .bool_set:N = \l_@@_dotted_bool ,
6151     dotted .initial:n = false ,
6152     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]`.

```

6153   color .code:n =
6154     \@@_set_CTarc:n { #1 }
6155     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6156   color .value_required:n = true ,
6157   sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6158   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.

```

6159   tikz .code:n =
6160     \IfPackageLoadedTF { tikz }
6161       { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6162       { \@@_error:n { tikz~without~tikz } } ,
6163   tikz .value_required:n = true ,
6164   total-width .dim_set:N = \l_@@_rule_width_dim ,
6165   total-width .value_required:n = true ,
6166   width .meta:n = { total-width = #1 } ,
6167   unknown .code:n = \@@_error:n { Unknown~key~for~RulesBis }
6168 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6169 \cs_new_protected:Npn \@@_vline:n #1
6170   {

```

The group is for the options.

```

6171   \group_begin:
6172   \int_set_eq:NN \l_@@_end_int \c@iRow
6173   \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).

```

6174   \int_compare:nNt { \l_@@_position_int } < { \c@jCol + 2 }
6175   \@@_vline_i:
6176   \group_end:
6177 }
6178 \cs_new_protected:Npn \@@_vline_i:
6179   {

```

`\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`.

```

6180   \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
6181   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int

```

```

6182 \l_tmpa_tl
6183 {

```

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.

```

6184 \bool_gset_true:N \g_tmpa_bool
6185 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6186 { \@@_test_vline_in_block:nnnnn ##1 }
6187 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6188 { \@@_test_vline_in_block:nnnnn ##1 }
6189 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6190 { \@@_test_vline_in_stroken_block:nnnn ##1 }
6191 \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_v: }
6192 \bool_if:NTF \g_tmpa_bool
6193 {
6194 \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.

```

6195 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6196 }
6197 {
6198 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6199 {
6200 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6201 \@@_vline_ii:
6202 \int_zero:N \l_@@_local_start_int
6203 }
6204 }
6205 }
6206 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6207 {
6208 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6209 \@@_vline_ii:
6210 }
6211 }

```

```

6212 \cs_new_protected:Npn \@@_test_in_corner_v:
6213 {
6214 \int_compare:nNnTF { \l_tmpb_tl } = { \c_jCol + 1 }
6215 {
6216 \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6217 { \bool_set_false:N \g_tmpa_bool }
6218 }
6219 {
6220 \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6221 {
6222 \int_compare:nNnTF { \l_tmpb_tl } = { \c_one_int }
6223 { \bool_set_false:N \g_tmpa_bool }
6224 {
6225 \@@_if_in_corner:nT
6226 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6227 { \bool_set_false:N \g_tmpa_bool }
6228 }
6229 }
6230 }
6231 }

```

```

6232 \cs_new_protected:Npn \@@_vline_ii:
6233 {

```

```

6234 \tl_clear:N \l_@@_tikz_rule_tl
6235 \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6236 \bool_if:NTF \l_@@_dotted_bool
6237 { \@@_vline_iv: }
6238 {
6239   \tl_if_empty:NTF \l_@@_tikz_rule_tl
6240   { \@@_vline_iii: }
6241   { \@@_vline_v: }
6242 }
6243 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

6244 \cs_new_protected:Npn \@@_vline_iii:
6245 {
6246   \pgfpicture
6247   \pgfrememberpicturepositiononpagetrue
6248   \pgf@relevantforpicturesizefalse
6249   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6250   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6251   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6252   \dim_set:Nn \l_tmpb_dim
6253   {
6254     \pgf@x
6255     - 0.5 \l_@@_rule_width_dim
6256     +
6257     ( \arrayrulewidth * \l_@@_multiplicity_int
6258       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6259   }
6260   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6261   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6262   \bool_lazy_all:nT
6263   {
6264     { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6265     { \cs_if_exist_p:N \CT@drsc@ }
6266     { ! \tl_if_blank_p:o \CT@drsc@ }
6267   }
6268   {
6269     \group_begin:
6270     \CT@drsc@
6271     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6272     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6273     \dim_set:Nn \l_@@_tmpd_dim
6274     {
6275       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6276       * ( \l_@@_multiplicity_int - 1 )
6277     }
6278     \pgfpathrectanglecorners
6279     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6280     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6281     \pgfusepath { fill }
6282     \group_end:
6283   }
6284   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6285   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6286   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6287   {
6288     \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6289     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6290     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6291   }
6292   \CT@arc@
6293   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6294   \pgfsetrectcap

```

```

6295 \pgfusepathqstroke
6296 \endpgfpicture
6297 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6298 \cs_new_protected:Npn \@@_vline_iv:
6299 {
6300   \pgfpicture
6301   \pgfrememberpicturepositiononpagetrue
6302   \pgf@relevantforpicturesizefalse
6303   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6304   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6305   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6306   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6307   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6308   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6309   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6310   \CT@arc@
6311   \@@_draw_line:
6312   \endpgfpicture
6313 }

```

The following code is for the case when the user uses the key `tikz`.

```

6314 \cs_new_protected:Npn \@@_vline_v:
6315 {
6316   \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).

```

6317   \CT@arc@
6318   \tl_if_empty:NF \l_@@_rule_color_tl
6319   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6320   \pgfrememberpicturepositiononpagetrue
6321   \pgf@relevantforpicturesizefalse
6322   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6323   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6324   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6325   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6326   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6327   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6328   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6329   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6330   ( \l_tmpb_dim , \l_tmpa_dim ) --
6331   ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6332   \end { tikzpicture }
6333 }

```

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).

```

6334 \cs_new_protected:Npn \@@_draw_vlines:
6335 {
6336   \int_step_inline:nnn
6337   {
6338     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6339     { 2 }
6340     { 1 }
6341   }
6342   {
6343     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6344     { \c@jCol }
6345     { \int_eval:n { \c@jCol + 1 } }

```

```

6346     }
6347     {
6348         \str_if_eq:eeF \l_@@_vlines_clist { all }
6349         { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6350         { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6351     }
6352 }

```

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}`.

```

6353 \cs_new_protected:Npn \@@_hline:n #1
6354 {

```

The group is for the options.

```

6355     \group_begin:
6356     \int_set_eq:NN \l_@@_end_int \c@jCol
6357     \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6358     \@@_hline_i:
6359     \group_end:
6360 }

6361 \cs_new_protected:Npn \@@_hline_i:
6362 {
6363     % \int_zero:N \l_@@_local_start_int
6364     % \int_zero:N \l_@@_local_end_int

```

`\l_tmpa_tl` is the number of row and `\l_tmpb_tl` the number of column. When we have found a column corresponding to a rule to draw, we note its number in `\l_@@_tmpc_tl`.

```

6365     \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6366     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6367     \l_tmpb_tl
6368     {

```

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.

```

6369         \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6370         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6371         { \@@_test_hline_in_block:nnnnn ##1 }

6372         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6373         { \@@_test_hline_in_block:nnnnn ##1 }

6374         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6375         { \@@_test_hline_in_stroken_block:nnnn ##1 }

6376         \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_h: }
6377         \bool_if:NTF \g_tmpa_bool
6378         {
6379             \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.

```

6380             { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6381         }
6382         {
6383             \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6384             {
6385                 \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6386                 \@@_hline_ii:
6387                 \int_zero:N \l_@@_local_start_int

```



```

6388     }
6389   }
6390 }
6391 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6392 {
6393   \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6394   \@@_hline_ii:
6395 }
6396 }

6397 \cs_new_protected:Npn \@@_test_in_corner_h:
6398 {
6399   \int_compare:nNnTF { \l_tmpa_tl } = { \c_iRow + 1 }
6400   {
6401     \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6402     { \bool_set_false:N \g_tmpa_bool }
6403   }
6404   {
6405     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6406     {
6407       \int_compare:nNnTF { \l_tmpa_tl } = { \c_one_int }
6408       { \bool_set_false:N \g_tmpa_bool }
6409       {
6410         \@@_if_in_corner:nT
6411         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6412         { \bool_set_false:N \g_tmpa_bool }
6413       }
6414     }
6415   }
6416 }

6417 \cs_new_protected:Npn \@@_hline_ii:
6418 {
6419   \tl_clear:N \l_@@_tikz_rule_tl
6420   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6421   \bool_if:NTF \l_@@_dotted_bool
6422   { \@@_hline_iv: }
6423   {
6424     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6425     { \@@_hline_iii: }
6426     { \@@_hline_v: }
6427   }
6428 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6429 \cs_new_protected:Npn \@@_hline_iii:
6430 {
6431   \pgfpicture
6432   \pgfrememberpicturepositiononpagetrue
6433   \pgf@relevantforpicturesizefalse
6434   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6435   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6436   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6437   \dim_set:Nn \l_tmpb_dim
6438   {
6439     \pgf@y
6440     - 0.5 \l_@@_rule_width_dim
6441     +
6442     ( \arrayrulewidth * \l_@@_multiplicity_int
6443       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6444   }

```

```

6445 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6446 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6447 \bool_lazy_all:nT
6448 {
6449   { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6450   { \cs_if_exist_p:N \CT@drsc@ }
6451   { ! \tl_if_blank_p:o \CT@drsc@ }
6452 }
6453 {
6454   \group_begin:
6455   \CT@drsc@
6456   \dim_set:Nn \l_@@_tmpd_dim
6457   {
6458     \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6459     * ( \l_@@_multiplicity_int - 1 )
6460   }
6461   \pgfpathrectanglecorners
6462   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6463   { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6464   \pgfusepathqfill
6465   \group_end:
6466 }
6467 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6468 \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6469 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6470 {
6471   \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6472   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6473   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6474 }
6475 \CT@arc@
6476 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6477 \pgfsetrectcap
6478 \pgfusepathqstroke
6479 \endpgfpicture
6480 }

```

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}
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```
6481 \cs_new_protected:Npn \@@_hline_iv:
```

```
6482 {
```

```
6483   \pgfpicture
```

```
6484   \pgfrememberpicturepositiononpagetrue
```

```
6485   \pgf@relevantforpicturesizefalse
```

```
6486   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
```

```
6487   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
```

```

6488 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6489 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6490 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6491 \int_compare:nNt { \l_@@_local_start_int } = { \c_one_int }
6492 {
6493   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6494   \bool_if:NF \g_@@_delims_bool
6495   { \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.

```

6496   \tl_if_eq:NnF \g_@@_left_delim_tl (
6497     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6498   )
6499 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6500 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6501 \int_compare:nNt { \l_@@_local_end_int } = { \c_jCol }
6502 {
6503   \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6504   \bool_if:NF \g_@@_delims_bool
6505   { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6506   \tl_if_eq:NnF \g_@@_right_delim_tl )
6507   { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6508 }
6509 \CT@arc@
6510 \@@_draw_line:
6511 \endpgfpicture
6512 }

```

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`).

```

6513 \cs_new_protected:Npn \@@_hline_v:
6514 {
6515   \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).

```

6516   \CT@arc@
6517   \tl_if_empty:NF \l_@@_rule_color_tl
6518   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6519   \pgfrememberpicturepositiononpagetrue
6520   \pgf@relevantforpicturesizefalse
6521   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6522   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6523   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6524   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6525   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6526   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6527   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6528   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6529     ( \l_tmpa_dim , \l_tmpb_dim ) --
6530     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6531   \end { tikzpicture }
6532 }

```

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).

```

6533 \cs_new_protected:Npn \@@_draw_hlines:
6534 {
6535   \int_step_inline:nnn

```

```

6536 { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6537 {
6538   \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6539   { \c@iRow }
6540   { \int_eval:n { \c@iRow + 1 } }
6541 }
6542 {
6543   \str_if_eq:eeF \l_@@_hlines_clist { all }
6544   { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6545   { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6546 }
6547 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6548 \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.

```

6549 \cs_set:Npn \@@_Hline_i:n #1
6550 {
6551   \peek_remove_spaces:n
6552   {
6553     \peek_meaning:NTF \Hline
6554     { \@@_Hline_ii:nn { #1 + 1 } }
6555     { \@@_Hline_iii:n { #1 } }
6556   }
6557 }
6558 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6559 \cs_set:Npn \@@_Hline_iii:n #1
6560 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6561 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6562 {
6563   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6564   \skip_vertical:N \l_@@_rule_width_dim
6565   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6566   {
6567     \@@_hline:n
6568     {
6569       multiplicity = #1 ,
6570       position = \int_eval:n { \c@iRow + 1 } ,
6571       total-width = \dim_use:N \l_@@_rule_width_dim ,
6572       #2
6573     }
6574   }
6575   \egroup
6576 }

```

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`).

```

6577 \cs_new_protected:Npn \@@_custom_line:n #1
6578 {
6579   \str_clear_new:N \l_@@_command_str
6580   \str_clear_new:N \l_@@_ccommand_str
6581   \str_clear_new:N \l_@@_letter_str
6582   \tl_clear_new:N \l_@@_other_keys_tl
6583   \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6584 \bool_lazy_all:nTF
6585 {
6586   { \str_if_empty_p:N \l_@@_letter_str }
6587   { \str_if_empty_p:N \l_@@_command_str }
6588   { \str_if_empty_p:N \l_@@_ccommand_str }
6589 }
6590 { \@@_error:n { No~letter~and~no~command } }
6591 { \@@_custom_line_i:o \l_@@_other_keys_tl }
6592 }
6593 \keys_define:nn { nicematrix / custom-line }
6594 {
6595   letter .str_set:N = \l_@@_letter_str ,
6596   letter .value_required:n = true ,
6597   command .str_set:N = \l_@@_command_str ,
6598   command .value_required:n = true ,
6599   ccommand .str_set:N = \l_@@_ccommand_str ,
6600   ccommand .value_required:n = true ,
6601 }

```

```

6602 \cs_new_protected:Npn \@@_custom_line_i:n #1
6603 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6604 \bool_set_false:N \l_@@_tikz_rule_bool
6605 \bool_set_false:N \l_@@_dotted_rule_bool
6606 \bool_set_false:N \l_@@_color_bool
6607 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6608 \bool_if:NT \l_@@_tikz_rule_bool
6609 {
6610   \IfPackageLoadedF { tikz }
6611   { \@@_error:n { tikz~in~custom~line~without~tikz } }
6612   \bool_if:NT \l_@@_color_bool
6613   { \@@_error:n { color~in~custom~line~with~tikz } }
6614 }
6615 \bool_if:NT \l_@@_dotted_rule_bool
6616 {
6617   \int_compare:nNnT { \l_@@_multiplicity_int } > { \c_one_int }
6618   { \@@_error:n { key~multiplicity~with~dotted } }
6619 }
6620 \str_if_empty:NF \l_@@_letter_str
6621 {
6622   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6623   { \@@_error:n { Several~letters } }
6624   {
6625     \tl_if_in:NoTF
6626     \c_@@_forbidden_letters_str
6627     \l_@@_letter_str
6628     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6629   }

```

During the analyse of the preamble provided by the final user, our automaton, for the letter corresponding at the custom line, will directly use the following command that you define in the main hash table of TeX.

```

6630 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6631 { \@@_v_custom_line:n { #1 } }
6632 }

```

```

6633     }
6634   }
6635   \str_if_empty:NF \l_@@_command_str { \l_@@_h_custom_line:n { #1 } }
6636   \str_if_empty:NF \l_@@_ccommand_str { \l_@@_c_custom_line:n { #1 } }
6637 }
6638 \cs_generate_variant:Nn \l_@@_custom_line_i:n { o }
6639 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6640 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\l_@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{nicematrix/Rules}`). That's why the following set of keys has some keys which are no-op.

```

6641 \keys_define:nn { nicematrix / custom-line-bis }
6642 {
6643   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6644   multiplicity .initial:n = 1 ,
6645   multiplicity .value_required:n = true ,
6646   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6647   color .value_required:n = true ,
6648   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6649   tikz .value_required:n = true ,
6650   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6651   dotted .value_forbidden:n = true ,
6652   total-width .code:n = { } ,
6653   total-width .value_required:n = true ,
6654   width .code:n = { } ,
6655   width .value_required:n = true ,
6656   sep-color .code:n = { } ,
6657   sep-color .value_required:n = true ,
6658   unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6659 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6660 \bool_new:N \l_@@_dotted_rule_bool
6661 \bool_new:N \l_@@_tikz_rule_bool
6662 \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`.

```

6663 \keys_define:nn { nicematrix / custom-line-width }
6664 {
6665   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6666   multiplicity .initial:n = 1 ,
6667   multiplicity .value_required:n = true ,
6668   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6669   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6670                       \bool_set_true:N \l_@@_total_width_bool ,
6671   total-width .value_required:n = true ,
6672   width .meta:n = { total-width = #1 } ,
6673   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6674 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule (hence the 'h' in the name) with the full width of the array. `#1` is the whole set of keys to pass to the command `\l_@@_hline:n` (which is in the internal `\CodeAfter`).

```

6675 \cs_new_protected:Npn \l_@@_h_custom_line:n #1
6676 {

```

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`).

```

6677 \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6678 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6679 }

```

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`).

```

6680 \cs_new_protected:Npn \@@_c_custom_line:n #1
6681 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

6682 \exp_args:Nc \NewExpandableDocumentCommand
6683 { nicematrix - \l_@@_ccommand_str }
6684 { 0 { } m }
6685 {
6686 \noalign
6687 {
6688 \@@_compute_rule_width:n { #1 , ##1 }
6689 \skip_vertical:n { \l_@@_rule_width_dim }
6690 \clist_map_inline:nn
6691 { ##2 }
6692 { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6693 }
6694 }
6695 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6696 }

```

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*.

```

6697 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6698 {
6699 \tl_if_in:nnTF { #2 } { - }
6700 { \@@_cut_on_hyphen:w #2 \q_stop }
6701 { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6702 \tl_gput_right:Ne \g_@@_pre_code_after_tl
6703 {
6704 \@@_hline:n
6705 {
6706 #1 ,
6707 start = \l_tmpa_tl ,
6708 end = \l_tmpb_tl ,
6709 position = \int_eval:n { \c@iRow + 1 } ,
6710 total-width = \dim_use:N \l_@@_rule_width_dim
6711 }
6712 }
6713 }

6714 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6715 {
6716 \bool_set_false:N \l_@@_tikz_rule_bool
6717 \bool_set_false:N \l_@@_total_width_bool
6718 \bool_set_false:N \l_@@_dotted_rule_bool
6719 \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6720 \bool_if:NF \l_@@_total_width_bool
6721 {
6722 \bool_if:NTF \l_@@_dotted_rule_bool
6723 { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6724 {
6725 \bool_if:NF \l_@@_tikz_rule_bool
6726 {

```

```

6727         \dim_set:Nn \l_@@_rule_width_dim
6728         {
6729             \arrayrulewidth * \l_@@_multiplicity_int
6730             + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6731         }
6732     }
6733 }
6734 }
6735 }
6736 \cs_new_protected:Npn \@@_v_custom_line:n #1
6737 {
6738     \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6739     \tl_gput_right:Ne \g_@@_array_preamble_tl
6740     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6741     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6742     {
6743         \@@_vline:n
6744         {
6745             #1 ,
6746             position = \int_eval:n { \c@jCol + 1 } ,
6747             total-width = \dim_use:N \l_@@_rule_width_dim
6748         }
6749     }
6750     \@@_rec_preamble:n
6751 }
6752 \@@_custom_line:n
6753 { 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.

```

6754 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6755 {
6756     \int_compare:nNnT { \l_tmpa_tl } > { #1 }
6757     {
6758         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6759         {
6760             \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6761             {
6762                 \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6763                 { \bool_gset_false:N \g_tmpa_bool }
6764             }
6765         }
6766     }
6767 }

```

The same for vertical rules.

```

6768 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6769 {
6770     \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6771     {
6772         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6773         {
6774             \int_compare:nNnT { \l_tmpb_tl } > { #2 }
6775             {
6776                 \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6777                 { \bool_gset_false:N \g_tmpa_bool }
6778             }
6779         }
6780     }
6781 }

```



```

6779     }
6780   }
6781 }
6782 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6783 {
6784   \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6785   {
6786     \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6787     {
6788       \int_compare:nNnTF { \l_tmpa_tl } = { #1 }
6789       { \bool_gset_false:N \g_tmpa_bool }
6790       {
6791         \int_compare:nNnT { \l_tmpa_tl } = { #3 + 1 }
6792         { \bool_gset_false:N \g_tmpa_bool }
6793       }
6794     }
6795   }
6796 }
6797 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6798 {
6799   \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6800   {
6801     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6802     {
6803       \int_compare:nNnTF { \l_tmpb_tl } = { #2 }
6804       { \bool_gset_false:N \g_tmpa_bool }
6805       {
6806         \int_compare:nNnT { \l_tmpb_tl } = { #4 + 1 }
6807         { \bool_gset_false:N \g_tmpa_bool }
6808       }
6809     }
6810   }
6811 }

```

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.

```

6812 \cs_new_protected:Npn \@@_compute_corners:
6813 {
6814   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6815   { \@@_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`).

```

6816   \clist_clear:N \l_@@_corners_cells_clist
6817   \clist_map_inline:Nn \l_@@_corners_clist
6818   {
6819     \str_case:nnF { ##1 }
6820     {
6821       { NW }
6822       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6823       { NE }
6824       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6825       { SW }
6826       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }

```

```

6827         { SE }
6828         { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6829     }
6830     { \@@_error:nn { bad~corner } { ##1 } }
6831 }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6832     \clist_if_empty:NF \l_@@_corners_cells_clist
6833     {

```

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.

```

6834         \tl_gput_right:Ne \g_@@_aux_tl
6835         {
6836             \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
6837             { \l_@@_corners_cells_clist }
6838         }
6839     }
6840 }

```

```

6841 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6842 {
6843     \int_step_inline:nnn { #1 } { #3 }
6844     {
6845         \int_step_inline:nnn { #2 } { #4 }
6846         { \cs_set_nopar:cpn { @@ _ block _ ##1 - ####1 } { } }
6847     }
6848 }

```

```

6849 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6850 {
6851     \cs_if_exist:cTF
6852     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6853     { \prg_return_true: }
6854     { \prg_return_false: }
6855 }

```

“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.

```

6856 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6857 {

```

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.

```

6858     \bool_set_false:N \l_tmpa_bool
6859     \int_zero_new:N \l_@@_last_empty_row_int
6860     \int_set:Nn \l_@@_last_empty_row_int { #1 }
6861     \int_step_inline:nnnn { #1 } { #3 } { #5 }

```

```

6862 {
6863   \bool_lazy_or:nnTF
6864   {
6865     \cs_if_exist_p:c
6866     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6867   }
6868   { \@@_if_in_block_p:nn { ##1 } { #2 } }
6869   { \bool_set_true:N \l_tmpa_bool }
6870   {
6871     \bool_if:NF \l_tmpa_bool
6872     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6873   }
6874 }

```

Now, you determine the last empty cell in the row of number 1.

```

6875 \bool_set_false:N \l_tmpa_bool
6876 \int_zero_new:N \l_@@_last_empty_column_int
6877 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6878 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6879 {
6880   \bool_lazy_or:nnTF
6881   {
6882     \cs_if_exist_p:c
6883     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6884   }
6885   { \@@_if_in_block_p:nn { #1 } { ##1 } }
6886   { \bool_set_true:N \l_tmpa_bool }
6887   {
6888     \bool_if:NF \l_tmpa_bool
6889     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6890   }
6891 }

```

Now, we loop over the rows.

```

6892 \int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int }
6893 {

```

We treat the row number ##1 with another loop.

```

6894   \bool_set_false:N \l_tmpa_bool
6895   \int_step_inline:nnnn { #2 } { #4 } { \l_@@_last_empty_column_int }
6896   {
6897     \bool_lazy_or:nnTF
6898     { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6899     { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6900     { \bool_set_true:N \l_tmpa_bool }
6901     {
6902       \bool_if:NF \l_tmpa_bool
6903       {
6904         \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6905         \clist_put_right:Nn
6906         \l_@@_corners_cells_clist
6907         { ##1 - #####1 }
6908         \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6909       }
6910     }
6911   }
6912 }
6913 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6914 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6915 \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:NeT \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.

```
6916 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```
6917 \keys_define:nn { nicematrix / NiceMatrixBlock }
6918 {
6919   auto-columns-width .code:n =
6920   {
6921     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6922     \dim_gzero_new:N \g_@@_max_cell_width_dim
6923     \bool_set_true:N \l_@@_auto_columns_width_bool
6924   }
6925 }

6926 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6927 {
6928   \int_gincr:N \g_@@_NiceMatrixBlock_int
6929   \dim_zero:N \l_@@_columns_width_dim
6930   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6931   \bool_if:NT \l_@@_block_auto_columns_width_bool
6932   {
6933     \cs_if_exist:cT
6934     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6935     {
6936       \dim_set:Nn \l_@@_columns_width_dim
6937       {
6938         \use:c
6939         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6940       }
6941     }
6942   }
6943 }
```

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).

```
6944 {
6945   \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.

```
6946   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6947   {
6948     \bool_if:NT \l_@@_block_auto_columns_width_bool
6949     {
6950       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6951       \iow_shipout:Ne \@mainaux
6952       {
6953         \cs_gset:cpn
6954         { @@ _ 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.

```
6955         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6956       }
6957       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6958     }
6959   }
6960   \ignorespacesafterend
6961 }
```

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).

```

6962 \cs_new_protected:Npn \@@_create_extra_nodes:
6963 {
6964   \bool_if:nTF \l_@@_medium_nodes_bool
6965   {
6966     \bool_if:NTF \l_@@_no_cell_nodes_bool
6967     { \@@_error:n { extra-nodes-with-no-cell-nodes } }
6968     {
6969       \bool_if:NTF \l_@@_large_nodes_bool
6970       \@@_create_medium_and_large_nodes:
6971       \@@_create_medium_nodes:
6972     }
6973   }
6974   {
6975     \bool_if:NT \l_@@_large_nodes_bool
6976     {
6977       \bool_if:NTF \l_@@_no_cell_nodes_bool
6978       { \@@_error:n { extra-nodes-with-no-cell-nodes } }
6979       \@@_create_large_nodes:
6980     }
6981   }
6982 }

```

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`.

```

6983 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6984 {
6985   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6986   {
6987     \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
6988     \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
6989     \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
6990     \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
6991   }
6992   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6993   {
6994     \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
6995     \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
6996     \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
6997     \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
6998   }

```

We begin the two nested loops over the rows and the columns of the array.

```

6999   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7000   {
7001       \int_step_variable:nnNn
7002       \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.

```

7003       {
7004           \cs_if_exist:cT
7005           { 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`.

```

7006       {
7007           \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7008           \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7009           { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7010           \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7011           {
7012               \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7013               { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7014           }

```

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`.

```

7015           \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7016           \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7017           { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } { \pgf@y } }
7018           \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7019           {
7020               \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7021               { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } { \pgf@x } }
7022           }
7023       }
7024   }
7025 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7026   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7027   {
7028       \dim_compare:nNnT
7029       { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7030       {
7031           \@@_qpoint:n { row - \@@_i: - base }
7032           \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7033           \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7034       }
7035   }
7036   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7037   {
7038       \dim_compare:nNnT
7039       { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7040       {
7041           \@@_qpoint:n { col - \@@_j: }
7042           \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7043           \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7044       }
7045   }
7046 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7047 \cs_new_protected:Npn \@@_create_medium_nodes:
7048 {
7049   \pgfpicture
7050     \pgfrememberpicturepositiononpagetrue
7051     \pgf@relevantforpicturesizefalse
7052     \@@_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”.

```

7053     \tl_set:Nn \l_@@_suffix_tl { -medium }
7054     \@@_create_nodes:
7055   \endpgfpicture
7056 }

```

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:`.

```

7057 \cs_new_protected:Npn \@@_create_large_nodes:
7058 {
7059   \pgfpicture
7060     \pgfrememberpicturepositiononpagetrue
7061     \pgf@relevantforpicturesizefalse
7062     \@@_computations_for_medium_nodes:
7063     \@@_computations_for_large_nodes:
7064     \tl_set:Nn \l_@@_suffix_tl { - large }
7065     \@@_create_nodes:
7066   \endpgfpicture
7067 }

7068 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7069 {
7070   \pgfpicture
7071     \pgfrememberpicturepositiononpagetrue
7072     \pgf@relevantforpicturesizefalse
7073     \@@_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”.

```

7074     \tl_set:Nn \l_@@_suffix_tl { - medium }
7075     \@@_create_nodes:
7076     \@@_computations_for_large_nodes:
7077     \tl_set:Nn \l_@@_suffix_tl { - large }
7078     \@@_create_nodes:
7079   \endpgfpicture
7080 }

```

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.

```

7081 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7082 {
7083   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7084   \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`.

```

7085   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7086   {
7087     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }

```

¹⁵If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7088     {
7089     (
7090         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7091         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7092     )
7093     / 2
7094     }
7095     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7096     { l_@@_row _ \@@_i: _ min_dim }
7097 }
7098 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7099 {
7100     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7101     {
7102     (
7103         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7104         \dim_use:c
7105         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7106     )
7107     / 2
7108     }
7109     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7110     { l_@@_column _ \@@_j: _ max _ dim }
7111 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7112     \dim_sub:cn
7113     { l_@@_column _ 1 _ min _ dim }
7114     \l_@@_left_margin_dim
7115     \dim_add:cn
7116     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7117     \l_@@_right_margin_dim
7118 }

```

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).

```

7119 \cs_new_protected:Npn \@@_create_nodes:
7120 {
7121     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7122     {
7123         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7124         {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7125         \@@_pgf_rect_node:nnnnn
7126         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7127         { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7128         { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7129         { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7130         { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7131         \str_if_empty:NF \l_@@_name_str
7132         {
7133             \pgfnodealias
7134             { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7135             { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7136         }
7137     }
7138 }
7139 \int_step_inline:nn { \c@iRow }

```



```

7140 {
7141   \pgfnodealias
7142   { \@@_env: - ##1 - last \l_@@_suffix_tl }
7143   { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7144 }
7145 \int_step_inline:nn { \c@jCol }
7146 {
7147   \pgfnodealias
7148   { \@@_env: - last - ##1 \l_@@_suffix_tl }
7149   { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7150 }
7151 \pgfnodealias % added 2025-04-05
7152 { \@@_env: - last - last \l_@@_suffix_tl }
7153 { \@@_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 .

```

7154   \seq_map_pairwise_function:NNN
7155   \g_@@_multicolumn_cells_seq
7156   \g_@@_multicolumn_sizes_seq
7157   \@@_node_for_multicolumn:nn
7158 }

7159 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7160 {
7161   \cs_set_nopar:Npn \@@_i: { #1 }
7162   \cs_set_nopar:Npn \@@_j: { #2 }
7163 }

```

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”).

```

7164 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7165 {
7166   \@@_extract_coords_values: #1 \q_stop
7167   \@@_pgf_rect_node:nnnnn
7168   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7169   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7170   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7171   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7172   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7173   \str_if_empty:NF \l_@@_name_str
7174   {
7175     \pgfnodealias
7176     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7177     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7178   }
7179 }

```

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).

```

7180 \keys_define:nn { nicematrix / Block / FirstPass }
7181 {
7182   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7183     \bool_set_true:N \l_@@_p_block_bool ,
7184   j .value_forbidden:n = true ,
7185   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7186   l .value_forbidden:n = true ,
7187   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7188   r .value_forbidden:n = true ,
7189   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7190   c .value_forbidden:n = true ,
7191   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7192   L .value_forbidden:n = true ,
7193   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7194   R .value_forbidden:n = true ,
7195   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7196   C .value_forbidden:n = true ,
7197   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7198   t .value_forbidden:n = true ,
7199   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7200   T .value_forbidden:n = true ,
7201   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7202   b .value_forbidden:n = true ,
7203   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7204   B .value_forbidden:n = true ,
7205   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7206   m .value_forbidden:n = true ,
7207   v-center .meta:n = m ,
7208   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7209   p .value_forbidden:n = true ,
7210   color .code:n =
7211     \@@_color:n { #1 }
7212     \tl_set_rescan:Nnn
7213       \l_@@_draw_tl
7214       { \char_set_catcode_other:N ! }
7215       { #1 } ,
7216   color .value_required:n = true ,
7217   respect-arraystretch .code:n =
7218     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7219   respect-arraystretch .value_forbidden:n = true ,
7220 }

```

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.

```

7221 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7222 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7223 {

```

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).

```

7224   \tl_if_blank:nTF { #2 }
7225   { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7226   {
7227     \tl_if_in:nnTF { #2 } { - }
7228     {
7229       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7230       \@@_Block_i_czech:w \@@_Block_i:w
7231       #2 \q_stop
7232     }
7233     {
7234       \@@_error:nn { Bad~argument~for~Block } { #2 }

```

```

7235         \@@_Block_ii:nnnnn \c_one_int \c_one_int
7236     }
7237 }
7238 { #1 } { #3 } { #4 }
7239 \ignorespaces
7240 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7241 \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`.

```

7242 {
7243   \char_set_catcode_active:N -
7244   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7245 }

```

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.

```

7246 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7247 {

```

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).

```

7248   \bool_lazy_or:nnTF
7249     { \tl_if_blank_p:n { #1 } }
7250     { \str_if_eq_p:ee { * } { #1 } }
7251     { \int_set:Nn \l_tmpa_int { 100 } }
7252     { \int_set:Nn \l_tmpa_int { #1 } }
7253   \bool_lazy_or:nnTF
7254     { \tl_if_blank_p:n { #2 } }
7255     { \str_if_eq_p:ee { * } { #2 } }
7256     { \int_set:Nn \l_tmpb_int { 100 } }
7257     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7258   \int_compare:nNnTF { \l_tmpb_int } = { \c_one_int }
7259   {
7260     \tl_if_empty:NTF \l_@@_hpos_cell_tl
7261       { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7262       { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7263   }
7264   { \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.

```

7265   \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7266   \tl_set:Ne \l_tmpa_tl
7267   {
7268     { \int_use:N \c@iRow }
7269     { \int_use:N \c@jCol }
7270     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7271     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7272   }

```

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).

```

7273 \bool_set_false:N \l_tmpa_bool
7274 \bool_if:NT \l_@@_amp_in_blocks_bool
\tl_if_in:nnT is slightly faster than \str_if_in:nnT.
7275 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7276 \bool_case:nF
7277 {
7278     \l_tmpa_bool { \@@_Block_vii:eennn }
7279     \l_@@_p_block_bool { \@@_Block_vi:eennn }

```

For the blocks mono-column, we will compose right now in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a `X` column, we should not do that since the width is determined by another way. This should be the same for the `p`, `m` and `b` columns and we should modify that point. However, for the `X` column, it’s imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

7280 \l_@@_X_bool { \@@_Block_v:eennn }
7281 { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eennn }
7282 { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7283 { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7284 }
7285 { \@@_Block_v:eennn }
7286 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7287 }

```

The following macro is for the case of a `\Block` which is mono-row or mono-column (or both) and don’t use the key `p`. In that case, the content of the block is composed right now in a box (because we have to take into account the dimensions of that box for the width of the current column or the height and the depth of the current row). However, that box will be put in the array *after the construction of the array* (by using PGF) with `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn` which will do the main job.

`#1` is *i* (the number of rows of the block), `#2` is *j* (the number of columns of the block), `#3` is the list of *key=values* pairs, `#4` are the tokens to put before the potential math mode and before the composition of the block and `#5` is the label (=content) of the block.

```

7288 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7289 {
7290     \int_gincr:N \g_@@_block_box_int
7291     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7292     {
7293         \tl_gput_right:Ne \g_@@_pre_code_after_tl
7294         {
7295             \@@_actually_diagbox:nnnnnn
7296             { \int_use:N \c@iRow }
7297             { \int_use:N \c@jCol }
7298             { \int_eval:n { \c@iRow + #1 - 1 } }
7299             { \int_eval:n { \c@jCol + #2 - 1 } }
7300             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7301             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7302         }
7303     }
7304     \box_gclear_new:c
7305     { 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!

```

7306 \hbox_gset:cn
7307 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7308 {

```

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`).

```

7309 \tl_if_empty:NTF \l_@@_color_tl
7310 { \int_compare:nNnT { #2 } = { \c_one_int } { \set@color } }
7311 { \@@_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`.

```

7312 \int_compare:nNnT { #1 } = { \c_one_int }
7313 {
7314 \int_if_zero:nTF { \c_iRow }
7315 {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7316 \cs_set_eq:NN \Block \@@_NullBlock:
7317 \l_@@_code_for_first_row_tl
7318 }
7319 {
7320 \int_compare:nNnT { \c_iRow } = { \l_@@_last_row_int }
7321 {
7322 \cs_set_eq:NN \Block \@@_NullBlock:
7323 \l_@@_code_for_last_row_tl
7324 }
7325 }
7326 \g_@@_row_style_tl
7327 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7328 \@@_reset_arraystretch:
7329 \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7330 #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.).

```

7331 \@@_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}`.

```

7332     \bool_if:NTF \l_@@_tabular_bool
7333     {
7334         \bool_lazy_all:nTF
7335         {
7336             { \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.

```

7337     {
7338         ! \dim_compare_p:nNn
7339         { \l_@@_col_width_dim } < { \c_zero_dim }
7340     }
7341     { ! \g_@@_rotate_bool }
7342 }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7343     {
7344         \use:e
7345         {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7346         \exp_not:N \begin { minipage }
7347         [ \str_lowercase:f \l_@@_vpos_block_str ]
7348         { \l_@@_col_width_dim }
7349         \str_case:on \l_@@_hpos_block_str
7350         { c \centering r \raggedleft l \raggedright }
7351     }
7352     #5
7353     \end { minipage }
7354 }

```

In the other cases, we use a `{tabular}`.

```

7355     {
7356         \use:e
7357         {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7358         \exp_not:N \begin { tabular }
7359         [ \str_lowercase:f \l_@@_vpos_block_str ]
7360         { @ { } \l_@@_hpos_block_str @ { } }
7361     }
7362     #5
7363     \end { tabular }
7364 }
7365 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is `false`). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7366     {
7367         \c_math_toggle_token
7368         \use:e
7369         {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7370         \exp_not:N \begin { array }
7371         [ \str_lowercase:f \l_@@_vpos_block_str ]
7372         { @ { } \l_@@_hpos_block_str @ { } }
7373     }
7374     #5
7375     \end { array }

```

```

7376         \c_math_toggle_token
7377     }
7378 }

```

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).

```

7379     \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.

```

7380     \int_compare:nNnT { #2 } = { \c_one_int }
7381     {
7382         \dim_gset:Nn \g_@@_blocks_wd_dim
7383         {
7384             \dim_max:nn
7385             { \g_@@_blocks_wd_dim }
7386             {
7387                 \box_wd:c
7388                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7389             }
7390         }
7391     }

```

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.

```

7392     \int_compare:nNnT { #1 } = { \c_one_int }
7393     {
7394         \bool_lazy_any:nT
7395         {
7396             { \str_if_empty_p:N \l_@@_vpos_block_str }
7397             { \str_if_eq_p:ee \l_@@_vpos_block_str { t } }
7398             { \str_if_eq_p:ee \l_@@_vpos_block_str { b } }
7399         }
7400         { \@@_adjust_blocks_ht_dp: }
7401     }
7402     \seq_gput_right:Ne \g_@@_blocks_seq
7403     {
7404         \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.

```

7405     {
7406         \exp_not:n { #3 } ,
7407         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7408     \bool_if:NT \g_@@_rotate_bool
7409     {
7410         \bool_if:NTF \g_@@_rotate_c_bool
7411         { m }
7412         {
7413             \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7414             { T }
7415         }
7416     }
7417 }
7418 {
7419     \box_use_drop:c
7420     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

```

7421     }
7422   }
7423   \bool_set_false:N \g_@@_rotate_c_bool
7424 }

7425 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7426 {
7427   \dim_gset:Nn \g_@@_blocks_ht_dim
7428   {
7429     \dim_max:nn
7430     { \g_@@_blocks_ht_dim }
7431     {
7432       \box_ht:c
7433       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7434     }
7435   }
7436   \dim_gset:Nn \g_@@_blocks_dp_dim
7437   {
7438     \dim_max:nn
7439     { \g_@@_blocks_dp_dim }
7440     {
7441       \box_dp:c
7442       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7443     }
7444   }
7445 }

7446 \cs_new:Npn \@@_adjust_hpos_rotate:
7447 {
7448   \bool_if:NT \g_@@_rotate_bool
7449   {
7450     \str_set:Ne \l_@@_hpos_block_str
7451     {
7452       \bool_if:NTF \g_@@_rotate_c_bool
7453       { c }
7454       {
7455         \str_case:onF \l_@@_vpos_block_str
7456         { b l B l t r T r }
7457         {
7458           \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
7459           { r }
7460           { l }
7461         }
7462       }
7463     }
7464   }
7465 }
7466 \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*.

```

7467 \cs_new_protected:Npn \@@_rotate_box_of_block:
7468 {
7469   \box_grotate:cn
7470   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7471   { 90 }
7472   \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7473   {
7474     \vbox_gset_top:cn
7475     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7476     {
7477       \skip_vertical:n { 0.8 ex }

```



```

7478         \box_use:c
7479         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7480     }
7481 }
7482 \bool_if:NT \g_@@_rotate_c_bool
7483 {
7484     \hbox_gset:cn
7485     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7486     {
7487         \c_math_toggle_token
7488         \vcenter
7489         {
7490             \box_use:c
7491             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7492         }
7493         \c_math_toggle_token
7494     }
7495 }
7496 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key p). In that case, the content of the block is *not* composed right now in a box. The composition in a box will be done further, just after the construction of the array (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn`).

#1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7497 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7498 {
7499     \seq_gput_right:Ne \g_@@_blocks_seq
7500     {
7501         \l_tmpa_tl
7502         { \exp_not:n { #3 } }
7503         {
7504             \bool_if:NTF \l_@@_tabular_bool
7505             {
7506                 \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7507         \@@_reset_arraystretch:
7508         \exp_not:n
7509         {
7510             \dim_zero:N \extrarowheight
7511             #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).

```

7512         \IfPackageLoadedTF { latex-lab-testphase-table }
7513         { \tag_stop:n { table } }
7514         \use:e
7515         {
7516             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7517             { @ { } \l_@@_hpos_block_str @ { } }
7518         }
7519         #5
7520         \end { tabular }
7521     }
7522     \group_end:
7523 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7524     {
7525         \group_begin:
The following will be no-op when respect-arraystretch is in force.
7526         \@@_reset_arraystretch:
7527         \exp_not:n
7528         {
7529             \dim_zero:N \extrarowheight
7530             #4
7531             \c_math_toggle_token
7532             \use:e
7533             {
7534                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7535                 { @ { } \l_@@_hpos_block_str @ { } }
7536             }
7537             #5
7538             \end { array }
7539             \c_math_toggle_token
7540         }
7541         \group_end:
7542     }
7543 }
7544 }
7545 }
7546 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7547 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7548 {
7549     \seq_gput_right:Ne \g_@@_blocks_seq
7550     {
7551         \l_tmpa_tl
7552         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7553         { { \exp_not:n { #4 #5 } } }
7554     }
7555 }
7556 \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`.

```

7557 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7558 {
7559     \seq_gput_right:Ne \g_@@_blocks_seq
7560     {
7561         \l_tmpa_tl
7562         { \exp_not:n { #3 } }
7563         { \exp_not:n { #4 #5 } }
7564     }
7565 }
7566 \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).

```

7567 \keys_define:nn { nicematrix / Block / SecondPass }
7568 {
7569     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7570     ampersand-in-blocks .default:n = true ,
7571     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7572   tikz .code:n =
7573       \IfPackageLoadedTF { tikz }
7574       { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7575       { \@@_error:n { tikz~key~without~tikz } } ,
7576   tikz .value_required:n = true ,
7577   fill .code:n =
7578       \tl_set_rescan:Nnn
7579       \l_@@_fill_tl
7580       { \char_set_catcode_other:N ! }
7581       { #1 } ,
7582   fill .value_required:n = true ,
7583   opacity .tl_set:N = \l_@@_opacity_tl ,
7584   opacity .value_required:n = true ,
7585   draw .code:n =
7586       \tl_set_rescan:Nnn
7587       \l_@@_draw_tl
7588       { \char_set_catcode_other:N ! }
7589       { #1 } ,
7590   draw .default:n = default ,
7591   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7592   rounded-corners .default:n = 4 pt ,
7593   color .code:n =
7594       \@@_color:n { #1 }
7595       \tl_set_rescan:Nnn
7596       \l_@@_draw_tl
7597       { \char_set_catcode_other:N ! }
7598       { #1 } ,
7599   borders .clist_set:N = \l_@@_borders_clist ,
7600   borders .value_required:n = true ,
7601   hvlines .meta:n = { vlines , hlines } ,
7602   vlines .bool_set:N = \l_@@_vlines_block_bool ,
7603   vlines .default:n = true ,
7604   hlines .bool_set:N = \l_@@_hlines_block_bool ,
7605   hlines .default:n = true ,
7606   line-width .dim_set:N = \l_@@_line_width_dim ,
7607   line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7608   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7609       \bool_set_true:N \l_@@_p_block_bool ,
7610   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7611   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7612   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7613   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7614       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7615   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7616       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7617   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7618       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7619   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7620   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7621   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7622   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7623   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7624   m .value_forbidden:n = true ,
7625   v-center .meta:n = m ,
7626   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7627   p .value_forbidden:n = true ,
7628   name .tl_set:N = \l_@@_block_name_str , % .str_set:N ?
7629   name .value_required:n = true ,
7630   name .initial:n = ,
7631   respect-arraystretch .code:n =
7632       \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,

```

```

7633     respect-arraystretch .value_forbidden:n = true ,
7634     transparent .bool_set:N = \l_@@_transparent_bool ,
7635     transparent .default:n = true ,
7636     transparent .initial:n = false ,
7637     unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7638 }

```

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.

```

7639 \cs_new_protected:Npn \@@_draw_blocks:
7640 {
7641   \bool_if:NTF \c_@@_revtex_bool
7642     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7643     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7644   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7645 }
7646 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7647 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7648   \int_zero:N \l_@@_last_row_int
7649   \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”).

```

7650   \int_compare:nNnTF { #3 } > { 98 }
7651     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7652     { \int_set:Nn \l_@@_last_row_int { #3 } }
7653   \int_compare:nNnTF { #4 } > { 98 }
7654     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7655     { \int_set:Nn \l_@@_last_col_int { #4 } }
7656   \int_compare:nNnTF { \l_@@_last_col_int } > { \g_@@_col_total_int }
7657     {
7658       \bool_lazy_and:nnTF
7659         { \l_@@_preamble_bool }
7660         {
7661           \int_compare_p:n
7662             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7663         }
7664       {
7665         \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7666         \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7667         \@@_msg_redirect_name:nn { columns-not-used } { none }
7668       }
7669       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7670     }
7671   {
7672     \int_compare:nNnTF { \l_@@_last_row_int } > { \g_@@_row_total_int }
7673       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7674       {
7675         \@@_Block_v:nneenn
7676           { #1 }
7677           { #2 }
7678           { \int_use:N \l_@@_last_row_int }
7679           { \int_use:N \l_@@_last_col_int }
7680           { #5 }

```

```

7681         { #6 }
7682     }
7683 }
7684 }

```

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

```

7685 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7686 {

```

The group is for the keys.

```

7687     \group_begin:
7688     \int_compare:nNtT { #1 } = { #3 }
7689     { \str_set:Nn \l_@@_vpos_block_str { t } }
7690     \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`.

```

7691     \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }

7692     \bool_lazy_and:nnT
7693     { \l_@@_vlines_block_bool }
7694     { ! \l_@@_ampersand_bool }
7695     {
7696         \tl_gput_right:Ne \g_nicematrix_code_after_tl
7697         {
7698             \@@_vlines_block:nnn
7699             { \exp_not:n { #5 } }
7700             { #1 - #2 }
7701             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7702         }
7703     }
7704     \bool_if:NT \l_@@_hlines_block_bool
7705     {
7706         \tl_gput_right:Ne \g_nicematrix_code_after_tl
7707         {
7708             \@@_hlines_block:nnn
7709             { \exp_not:n { #5 } }
7710             { #1 - #2 }
7711             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7712         }
7713     }
7714     \bool_if:NF \l_@@_transparent_bool
7715     {
7716         \bool_lazy_and:nnF { \l_@@_vlines_block_bool } { \l_@@_hlines_block_bool }
7717         {

```

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).

```

7718         \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7719         { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7720     }
7721 }

```

```

7722 \tl_if_empty:NF \l_@@_draw_tl
7723 {
7724     \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7725     { \@@_error:n { hlines-with-color } }
7726     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7727     {
7728         \@@_stroke_block:nnn

```

#5 are the options

```

7729         { \exp_not:n { #5 } }
7730         { #1 - #2 }
7731         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7732     }
7733     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7734     { { #1 } { #2 } { #3 } { #4 } }
7735 }
7736 \clist_if_empty:NF \l_@@_borders_clist
7737 {
7738     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7739     {
7740         \@@_stroke_borders_block:nnn
7741         { \exp_not:n { #5 } }
7742         { #1 - #2 }
7743         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7744     }
7745 }
7746 \tl_if_empty:NF \l_@@_fill_tl
7747 {
7748     \@@_add_opacity_to_fill:
7749     \tl_gput_right:Ne \g_@@_pre_code_before_tl
7750     {
7751         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7752         { #1 - #2 }
7753         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7754         { \dim_use:N \l_@@_rounded_corners_dim }
7755     }
7756 }
7757 \seq_if_empty:NF \l_@@_tikz_seq
7758 {
7759     \tl_gput_right:Ne \g_nicematrix_code_before_tl
7760     {
7761         \@@_block_tikz:nnnnn
7762         { \seq_use:Nn \l_@@_tikz_seq { , } }
7763         { #1 }
7764         { #2 }
7765         { \int_use:N \l_@@_last_row_int }
7766         { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7767     }
7768 }
7769 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7770 {
7771     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7772     {
7773         \@@_actually_diagbox:nnnnnn
7774         { #1 }
7775         { #2 }
7776         { \int_use:N \l_@@_last_row_int }
7777         { \int_use:N \l_@@_last_col_int }
7778         { \exp_not:n { ##1 } }
7779         { \exp_not:n { ##2 } }
7780     }
7781 }

```

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.

```

7782 \pgfpicture
7783 \pgfrememberpicturepositiononpagetrue
7784 \pgf@relevantforpicturesizefalse
7785 \@@_qpoint:n { row - #1 }
7786 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7787 \@@_qpoint:n { col - #2 }
7788 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7789 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7790 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7791 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7792 \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.

```

7793 \@@_pgf_rect_node:nnnnn
7794 { \@@_env: - #1 - #2 - block }
7795 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7796 \str_if_empty:NF \l_@@_block_name_str
7797 {
7798   \pgfnodealias
7799   { \@@_env: - \l_@@_block_name_str }
7800   { \@@_env: - #1 - #2 - block }
7801   \str_if_empty:NF \l_@@_name_str
7802   {
7803     \pgfnodealias
7804     { \l_@@_name_str - \l_@@_block_name_str }
7805     { \@@_env: - #1 - #2 - block }
7806   }
7807 }

```

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.

```

7808 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7809 {
7810   \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.

```

7811 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7812 {

```

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.

```

7813 \cs_if_exist:cT
7814 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }

```

```

7815     {
7816         \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7817         {
7818             \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7819             \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7820         }
7821     }
7822 }

```

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.

```

7823     \dim_compare:nNnT { \l_tmpb_dim } = { \c_max_dim }
7824     {
7825         \@@_qpoint:n { col - #2 }
7826         \dim_set_eq:NN \l_tmpb_dim \pgf@x
7827     }
7828     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7829     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7830     {
7831         \cs_if_exist:cT
7832         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7833         {
7834             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7835             {
7836                 \pgfpointanchor
7837                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7838                 { east }
7839                 \dim_set:Nn \l_@@_tmpd_dim
7840                 { \dim_max:nn { \l_@@_tmpd_dim } { \pgf@x } }
7841             }
7842         }
7843     }
7844     \dim_compare:nNnT { \l_@@_tmpd_dim } = { - \c_max_dim }
7845     {
7846         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7847         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7848     }
7849     \@@_pgf_rect_node:nnnnn
7850     { \@@_env: - #1 - #2 - block - short }
7851     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7852 }

```

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.

```

7853     \bool_if:NT \l_@@_medium_nodes_bool
7854     {
7855         \@@_pgf_rect_node:nnn
7856         { \@@_env: - #1 - #2 - block - medium }
7857         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7858         {
7859             \pgfpointanchor
7860             { \@@_env:
7861               - \int_use:N \l_@@_last_row_int
7862               - \int_use:N \l_@@_last_col_int - medium
7863             }
7864             { south-east }
7865         }
7866     }
7867     \endpgfpicture
7868
7869     \bool_if:NTF \l_@@_ampersand_bool
7870     {

```



```

7871 \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7872 \int_zero_new:N \l_@@_split_int
7873 \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7874 \pgfpicture
7875 \pgfrememberpicturepositiononpagetrue
7876 \pgf@relevantforpicturesizefalse
7877
7878 \@@_qpoint:n { row - #1 }
7879 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7880 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7881 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7882 \@@_qpoint:n { col - #2 }
7883 \dim_set_eq:NN \l_tmpa_dim \pgf@x
7884 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7885 \dim_set:Nn \l_tmpb_dim
7886 { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7887 \bool_lazy_or:nnT
7888 { \l_@@_vlines_block_bool }
7889 { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
7890 {
7891   \int_step_inline:nn { \l_@@_split_int - 1 }
7892   {
7893     \pgfpathmoveto
7894     {
7895       \pgfpoint
7896       { \l_tmpa_dim + ##1 \l_tmpb_dim }
7897       \l_@@_tmpc_dim
7898     }
7899     \pgfpathlineto
7900     {
7901       \pgfpoint
7902       { \l_tmpa_dim + ##1 \l_tmpb_dim }
7903       \l_@@_tmpd_dim
7904     }
7905     \CT@arc@
7906     \pgfsetlinewidth { 1.1 \arrayrulewidth }
7907     \pgfsetrectcap
7908     \pgfusepathqstroke
7909   }
7910 }
7911 \@@_qpoint:n { row - #1 - base }
7912 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7913 \int_step_inline:nn { \l_@@_split_int }
7914 {
7915   \group_begin:
7916   \dim_set:Nn \col@sep
7917   { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
7918   \pgftransformshift
7919   {
7920     \pgfpoint
7921     {
7922       \l_tmpa_dim + ##1 \l_tmpb_dim -
7923       \str_case:on \l_@@_hpos_block_str
7924       {
7925         l { \l_tmpb_dim + \col@sep }
7926         c { 0.5 \l_tmpb_dim }
7927         r { \col@sep }
7928       }
7929     }
7930     { \l_@@_tmpc_dim }
7931   }
7932   \pgfset { inner~sep = \c_zero_dim }
7933   \pgfnode

```

```

7934     { rectangle }
7935     {
7936       \str_case:on \l_@@_hpos_block_str
7937       {
7938         c { base }
7939         l { base~west }
7940         r { base~east }
7941       }
7942     }
7943     { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
7944     \group_end:
7945   }
7946   \endpgfpicture
7947 }

```

Now the case where there is no ampersand & in the content of the block.

```

7948 {
7949   \bool_if:NTF \l_@@_p_block_bool
7950   {

```

When the final user has used the key p, we have to compute the width.

```

7951     \pgfpicture
7952     \pgfrememberpicturepositiononpagetrue
7953     \pgf@relevantforpicturesizefalse
7954     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7955     {
7956       \@@_qpoint:n { col - #2 }
7957       \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7958       \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7959     }
7960     {
7961       \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7962       \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7963       \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7964     }
7965     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7966   \endpgfpicture
7967   \hbox_set:Nn \l_@@_cell_box
7968   {
7969     \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
7970     { \g_tmpb_dim }
7971     \str_case:on \l_@@_hpos_block_str
7972     { c \centering r \raggedleft l \raggedright j { } }
7973     #6
7974     \end { minipage }
7975   }
7976 }
7977 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7978 \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.

```

7979   \pgfpicture
7980   \pgfrememberpicturepositiononpagetrue
7981   \pgf@relevantforpicturesizefalse
7982   \bool_lazy_any:nTF
7983   {
7984     { \str_if_empty_p:N \l_@@_vpos_block_str }
7985     { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7986     { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7987     { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7988   }
7989   {

```

If we are in the first column, we must put the block as if it was with the key r.

```
7990 \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.

```
7991 \bool_if:nT \g_@@_last_col_found_bool
7992 {
7993   \int_compare:nNtT { #2 } = { \g_@@_col_total_int }
7994   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7995 }
```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```
7996 \tl_set:Ne \l_tmpa_tl
7997 {
7998   \str_case:on \l_@@_vpos_block_str
7999   {
```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```
8000   { } {
8001     \str_case:on \l_@@_hpos_block_str
8002     {
8003       c { center }
8004       l { west }
8005       r { east }
8006       j { center }
8007     }
8008   }
8009   c {
8010     \str_case:on \l_@@_hpos_block_str
8011     {
8012       c { center }
8013       l { west }
8014       r { east }
8015       j { center }
8016     }
8017   }
8018 }
8019 T {
8020   \str_case:on \l_@@_hpos_block_str
8021   {
8022     c { north }
8023     l { north-west }
8024     r { north-east }
8025     j { north }
8026   }
8027 }
8028 }
8029 B {
8030   \str_case:on \l_@@_hpos_block_str
8031   {
8032     c { south }
8033     l { south-west }
8034     r { south-east }
8035     j { south }
8036   }
8037 }
8038 }
8039 }
8040 }
8041 \pgftransformshift
8042 {
8043   \pgfpointanchor
8044   {
8045     \@@_env: - #1 - #2 - block
```

```

8046         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8047     }
8048     { \l_tmpa_tl }
8049 }
8050 \pgfset { inner~sep = \c_zero_dim }
8051 \pgfnode
8052 { rectangle }
8053 { \l_tmpa_tl }
8054 { \box_use_drop:N \l_@@_cell_box } { } { }
8055 }

```

End of the case when `\l_@@_vpos_block_str` is equal to c, T or B. Now, the other cases.

```

8056 {
8057     \pgfextracty \l_tmpa_dim
8058     {
8059         \@@_qpoint:n
8060         {
8061             row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
8062             - base
8063         }
8064     }
8065     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the *x*-value of the center of the block.

```

8066     \pgfpointanchor
8067     {
8068         \@@_env: - #1 - #2 - block
8069         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8070     }
8071     {
8072         \str_case:on \l_@@_hpos_block_str
8073         {
8074             c { center }
8075             l { west }
8076             r { east }
8077             j { center }
8078         }
8079     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8080     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8081     \pgfset { inner~sep = \c_zero_dim }
8082     \pgfnode
8083     { rectangle }
8084     {
8085         \str_case:on \l_@@_hpos_block_str
8086         {
8087             c { base }
8088             l { base~west }
8089             r { base~east }
8090             j { base }
8091         }
8092     }
8093     { \box_use_drop:N \l_@@_cell_box } { } { }
8094 }
8095 \endpgfpicture
8096 }
8097 \group_end:
8098 }
8099 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }

```

For the command `\cellcolor` used within a sub-cell of a `\Block` (when the character `&` is used inside the cell).

```

8100 \cs_set_protected:Npn \@@_fill:nnnnn #1 #2 #3 #4 #5
8101 {
8102   \pgfpicture
8103   \pgfrememberpicturepositiononpagetrue
8104   \pgf@relevantforpicturesizefalse
8105   \pgfpathrectanglecorners
8106     { \pgfpoint { #2 } { #3 } }
8107     { \pgfpoint { #4 } { #5 } }
8108   \pgfsetfillcolor { #1 }
8109   \pgfusepath { fill }
8110   \endpgfpicture
8111 }

```

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).

```

8112 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8113 {
8114   \tl_if_empty:NF \l_@@_opacity_tl
8115   {
8116     \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8117       {
8118         \tl_set:Nc \l_@@_fill_tl
8119           {
8120             [ opacity = \l_@@_opacity_tl ,
8121             \tl_tail:o \l_@@_fill_tl
8122           }
8123       }
8124       {
8125         \tl_set:Nc \l_@@_fill_tl
8126           { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8127       }
8128     }
8129   }

```

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).

```

8130 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8131 {
8132   \group_begin:
8133   \tl_clear:N \l_@@_draw_tl
8134   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8135   \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8136   \pgfpicture
8137   \pgfrememberpicturepositiononpagetrue
8138   \pgf@relevantforpicturesizefalse
8139   \tl_if_empty:NF \l_@@_draw_tl
8140   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8141     \tl_if_eq:NnTF \l_@@_draw_tl { default }
8142       { \CT@arc@ }
8143       { \@@_color:o \l_@@_draw_tl }
8144   }
8145   \pgfsetcornersarced
8146   {
8147     \pgfpoint
8148       { \l_@@_rounded_corners_dim }
8149       { \l_@@_rounded_corners_dim }
8150   }

```

```

8151 \@@_cut_on_hyphen:w #2 \q_stop
8152 \int_compare:nNnF { \l_tmpa_tl } > { \c@iRow }
8153 {
8154   \int_compare:nNnF { \l_tmpb_tl } > { \c@jCol }
8155   {
8156     \@@_qpoint:n { row - \l_tmpa_tl }
8157     \dim_set_eq:NN \l_tmpb_dim \pgf@y
8158     \@@_qpoint:n { col - \l_tmpb_tl }
8159     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8160     \@@_cut_on_hyphen:w #3 \q_stop
8161     \int_compare:nNnT { \l_tmpa_tl } > { \c@iRow }
8162     { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8163     \int_compare:nNnT { \l_tmpb_tl } > { \c@jCol }
8164     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8165     \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8166     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8167     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8168     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8169     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8170     \pgfpathrectanglecorners
8171     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8172     { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8173     \dim_compare:nNnTF { \l_@@_rounded_corners_dim } = { \c_zero_dim }
8174     { \pgfusepathqstroke }
8175     { \pgfusepath { stroke } }
8176   }
8177 }
8178 \endpgfpicture
8179 \group_end:
8180 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8181 \keys_define:nn { nicematrix / BlockStroke }
8182 {
8183   color .tl_set:N = \l_@@_draw_tl ,
8184   draw .code:n =
8185     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8186   draw .default:n = default ,
8187   line-width .dim_set:N = \l_@@_line_width_dim ,
8188   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8189   rounded-corners .default:n = 4 pt
8190 }

```

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).

```

8191 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8192 {
8193   \group_begin:
8194   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8195   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8196   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8197   \@@_cut_on_hyphen:w #2 \q_stop
8198   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8199   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8200   \@@_cut_on_hyphen:w #3 \q_stop
8201   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8202   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8203   \int_step_inline:nnn { \l_@@_tmpd_tl } { \l_tmpb_tl }
8204   {
8205     \use:e
8206     {
8207       \@@_vline:n

```

```

8208         {
8209             position = ##1 ,
8210             start = \l_@@_tmpc_tl ,
8211             end = \int_eval:n { \l_tmpa_tl - 1 } ,
8212             total-width = \dim_use:N \l_@@_line_width_dim
8213         }
8214     }
8215 }
8216 \group_end:
8217 }
8218 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8219 {
8220     \group_begin:
8221     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8222     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8223     \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8224     \@@_cut_on_hyphen:w #2 \q_stop
8225     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8226     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8227     \@@_cut_on_hyphen:w #3 \q_stop
8228     \tl_set:Nc \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8229     \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8230     \int_step_inline:nnn { \l_@@_tmpc_tl } { \l_tmpa_tl }
8231     {
8232         \use:e
8233         {
8234             \@@_hline:n
8235             {
8236                 position = ##1 ,
8237                 start = \l_@@_tmpd_tl ,
8238                 end = \int_eval:n { \l_tmpb_tl - 1 } ,
8239                 total-width = \dim_use:N \l_@@_line_width_dim
8240             }
8241         }
8242     }
8243     \group_end:
8244 }

```

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).

```

8245 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8246 {
8247     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8248     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8249     \dim_compare:nNnTF { \l_@@_rounded_corners_dim } > { \c_zero_dim }
8250     { \@@_error:n { borders~forbidden } }
8251     {
8252         \tl_clear_new:N \l_@@_borders_tikz_tl
8253         \keys_set:no
8254         { nicematrix / OnlyForTikzInBorders }
8255         \l_@@_borders_clist
8256         \@@_cut_on_hyphen:w #2 \q_stop
8257         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8258         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8259         \@@_cut_on_hyphen:w #3 \q_stop
8260         \tl_set:Nc \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8261         \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8262         \@@_stroke_borders_block_i:
8263     }
8264 }
8265 \hook_gput_code:nnn { begindocument } { . }

```

```

8266 {
8267   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8268   {
8269     \c_@@_pgfortikzpicture_tl
8270     \@@_stroke_borders_block_ii:
8271     \c_@@_endpgfortikzpicture_tl
8272   }
8273 }

8274 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8275 {
8276   \pgfrememberpicturepositiononpagetrue
8277   \pgf@relevantforpicturesizefalse
8278   \CT@arc@
8279   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8280   \clist_if_in:NnT \l_@@_borders_clist { right }
8281   { \@@_stroke_vertical:n \l_tmpb_tl }
8282   \clist_if_in:NnT \l_@@_borders_clist { left }
8283   { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8284   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8285   { \@@_stroke_horizontal:n \l_tmpa_tl }
8286   \clist_if_in:NnT \l_@@_borders_clist { top }
8287   { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8288 }

8289 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8290 {
8291   tikz .code:n =
8292     \cs_if_exist:NTF \tikzpicture
8293     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8294     { \@@_error:n { tikz~in~borders~without~tikz } } ,
8295   tikz .value_required:n = true ,
8296   top .code:n = ,
8297   bottom .code:n = ,
8298   left .code:n = ,
8299   right .code:n = ,
8300   unknown .code:n = \@@_error:n { bad~border }
8301 }

```

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).

```

8302 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8303 {
8304   \@@_qpoint:n \l_@@_tmpc_tl
8305   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8306   \@@_qpoint:n \l_tmpa_tl
8307   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8308   \@@_qpoint:n { #1 }
8309   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8310   {
8311     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8312     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8313     \pgfusepathqstroke
8314   }
8315   {
8316     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8317     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8318   }
8319 }

```

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).

```

8320 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8321 {

```



```

8322 \@@_qpoint:n \l_@@_tmpd_tl
8323 \clist_if_in:NnTF \l_@@_borders_clist { left }
8324   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8325   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8326 \@@_qpoint:n \l_tmpb_tl
8327 \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8328 \@@_qpoint:n { #1 }
8329 \tl_if_empty:NTF \l_@@_borders_tikz_tl
8330   {
8331     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8332     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8333     \pgfusepathqstroke
8334   }
8335   {
8336     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8337     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8338   }
8339 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

8340 \keys_define:nn { nicematrix / BlockBorders }
8341 {
8342   borders .clist_set:N = \l_@@_borders_clist ,
8343   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8344   rounded-corners .default:n = 4 pt ,
8345   line-width .dim_set:N = \l_@@_line_width_dim
8346 }

```

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.

```

8347 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8348 {
8349   \begin { tikzpicture }
8350   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because `#5` is a list of lists.

```

8351   \clist_map_inline:nn { #1 }
8352   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8353   \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8354   \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8355   (
8356     [
8357       xshift = \dim_use:N \l_@@_offset_dim ,
8358       yshift = - \dim_use:N \l_@@_offset_dim
8359     ]
8360     #2 -| #3
8361   )
8362   rectangle
8363   (
8364     [
8365       xshift = - \dim_use:N \l_@@_offset_dim ,
8366       yshift = \dim_use:N \l_@@_offset_dim
8367     ]
8368     \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8369   ) ;

```

```

8370     }
8371   \end { tikzpicture }
8372 }
8373 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

8374 \keys_define:nn { nicematrix / SpecialOffset }
8375 { 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.

```

8376 \cs_new_protected:Npn \@@_NullBlock:
8377 { \@@_collect_options:n { \@@_NullBlock_i: } }
8378 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8379 { }

```

27 How to draw the dotted lines transparently

```

8380 \cs_set_protected:Npn \@@_renew_matrix:
8381 {
8382   \RenewDocumentEnvironment { pmatrix } { } {
8383     { \pNiceMatrix }
8384     { \endpNiceMatrix }
8385   \RenewDocumentEnvironment { vmatrix } { } {
8386     { \vNiceMatrix }
8387     { \endvNiceMatrix }
8388   \RenewDocumentEnvironment { Vmatrix } { } {
8389     { \VNiceMatrix }
8390     { \endVNiceMatrix }
8391   \RenewDocumentEnvironment { bmatrix } { } {
8392     { \bNiceMatrix }
8393     { \endbNiceMatrix }
8394   \RenewDocumentEnvironment { Bmatrix } { } {
8395     { \BNiceMatrix }
8396     { \endBNiceMatrix }
8397 }

```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8398 \keys_define:nn { nicematrix / Auto }
8399 {
8400   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8401   columns-type .value_required:n = true ,
8402   l .meta:n = { columns-type = l } ,
8403   r .meta:n = { columns-type = r } ,
8404   c .meta:n = { columns-type = c } ,
8405   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8406   delimiters / color .value_required:n = true ,
8407   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8408   delimiters / max-width .default:n = true ,
8409   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8410   delimiters .value_required:n = true ,
8411   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8412   rounded-corners .default:n = 4 pt
8413 }

```

```

8414 \NewDocumentCommand \AutoNiceMatrixWithDelims
8415 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8416 { \@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8417 \cs_new_protected:Npn \@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8418 {

```

The group is for the protection of the keys.

```

8419 \group_begin:
8420 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8421 \use:e
8422 {
8423 \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8424 { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8425 [ \exp_not:o \l_tmpa_tl ]
8426 }
8427 \int_if_zero:nT { \l_@@_first_row_int }
8428 {
8429 \int_if_zero:nT { \l_@@_first_col_int } { & }
8430 \prg_replicate:nn { #4 - 1 } { & }
8431 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8432 }
8433 \prg_replicate:nn { #3 }
8434 {
8435 \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).

```

8436 \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8437 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8438 }
8439 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
8440 {
8441 \int_if_zero:nT { \l_@@_first_col_int } { & }
8442 \prg_replicate:nn { #4 - 1 } { & }
8443 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8444 }
8445 \end { NiceArrayWithDelims }
8446 \group_end:
8447 }

8448 \cs_set_protected:Npn \@_define_com:NNN #1 #2 #3
8449 {
8450 \cs_set_protected:cpn { #1 AutoNiceMatrix }
8451 {
8452 \bool_gset_true:N \g_@@_delims_bool
8453 \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8454 \AutoNiceMatrixWithDelims { #2 } { #3 }
8455 }
8456 }

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8457 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8458 {
8459 \group_begin:
8460 \bool_gset_false:N \g_@@_delims_bool
8461 \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8462 \group_end:
8463 }

```

29 The redefinition of the command `\dotfill`

```

8464 \cs_set_eq:NN \@@_old_dotfill: \dotfill
8465 \cs_new_protected:Npn \@@_dotfill:
8466 {

```

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}`).

```

8467   \@@_old_dotfill:
8468   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8469 }

```

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`.

```

8470 \cs_new_protected:Npn \@@_dotfill_i:
8471 {
8472   \dim_compare:nNtT { \box_wd:N \l_@@_cell_box } = { \c_zero_dim }
8473   { \@@_old_dotfill: }
8474 }

```

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.

```

8475 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8476 {
8477   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8478   {
8479     \@@_actually_diagbox:nnnnnn
8480     { \int_use:N \c@iRow }
8481     { \int_use:N \c@jCol }
8482     { \int_use:N \c@iRow }
8483     { \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.

```

8484     { \g_@@_row_style_tl \exp_not:n { #1 } }
8485     { \g_@@_row_style_tl \exp_not:n { #2 } }
8486   }

```

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`.

```

8487   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8488   {
8489     { \int_use:N \c@iRow }
8490     { \int_use:N \c@jCol }
8491     { \int_use:N \c@iRow }
8492     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8493     { }
8494   }
8495 }

```

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.

```

8496 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8497 {
8498   \pgfpicture
8499   \pgf@relevantforpicturesizefalse
8500   \pgfrememberpicturepositiononpagetrue
8501   \@@_qpoint:n { row - #1 }
8502   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8503   \@@_qpoint:n { col - #2 }
8504   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8505   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8506   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8507   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8508   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8509   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8510   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8511   {

```

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.

```

8512   \CT@arc@
8513   \pgfsetroundcap
8514   \pgfusepathqstroke
8515 }
8516 \pgfset { inner~sep = 1 pt }
8517 \pgfscope
8518 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8519 \pgfnode { rectangle } { south~west }
8520 {
8521   \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8522   \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8523   \end { minipage }
8524 }
8525 { }
8526 { }
8527 \endpgfscope
8528 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8529 \pgfnode { rectangle } { north~east }
8530 {
8531   \begin { minipage } { 20 cm }
8532   \raggedleft
8533   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8534   \end { minipage }
8535 }
8536 { }
8537 { }
8538 \endpgfpicture
8539 }

```

31 The keyword `\CodeAfter`

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment `{@@-light-syntax}` on p. 85.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
8540 \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 `\`.

```
8541 \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`.

```
8542 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8543 {
8544   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8545   \@@_CodeAfter_iv:n
8546 }
```

We catch the argument of the command `\end` (in `#1`).

```
8547 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8548 {
```

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.

```
8549   \str_if_eq:eeTF \@currenvir { #1 }
8550   { \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`.

```
8551   {
8552     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8553     \@@_CodeAfter_ii:n
8554   }
8555 }
```

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.

```
8556 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8557 {
8558   \pgfpicture
8559   \pgfrememberpicturepositiononpagetrue
8560   \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.

```
8561   \@@_qpoint:n { row - 1 }
8562   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8563   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8564   \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).

```

8565 \bool_if:nTF { #3 }
8566 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8567 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8568 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8569 {
8570   \cs_if_exist:cT
8571   { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8572   {
8573     \pgfpointanchor
8574     { \@@_env: - ##1 - #2 }
8575     { \bool_if:nTF { #3 } { west } { east } }
8576     \dim_set:Nn \l_tmpa_dim
8577     {
8578       \bool_if:nTF { #3 }
8579       { \dim_min:nn }
8580       { \dim_max:nn }
8581       \l_tmpa_dim
8582       { \pgf@x }
8583     }
8584   }
8585 }

```

Now we can put the delimiter with a node of PGF.

```

8586 \pgfset { inner~sep = \c_zero_dim }
8587 \dim_zero:N \nulldelimiterspace
8588 \pgftransformshift
8589 {
8590   \pgfpoint
8591   { \l_tmpa_dim }
8592   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8593 }
8594 \pgfnode
8595 { rectangle }
8596 { \bool_if:nTF { #3 } { east } { west } }
8597 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8598 \nullfont
8599 \c_math_toggle_token
8600 \@@_color:o \l_@@_delimiters_color_tl
8601 \bool_if:nTF { #3 } { \left #1 } { \left . }
8602 \vcenter
8603 {
8604   \nullfont
8605   \hrule \@@height
8606   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8607   \@@depth \c_zero_dim
8608   \@@width \c_zero_dim
8609 }
8610 \bool_if:nTF { #3 } { \right . } { \right #1 }
8611 \c_math_toggle_token
8612 }
8613 { }
8614 { }
8615 \endpgfpicture
8616 }

```

33 The command `\SubMatrix`

```

8617 \keys_define:nn { nicematrix / sub-matrix }
8618 {
8619   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8620   extra-height .value_required:n = true ,
8621   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8622   left-xshift .value_required:n = true ,
8623   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8624   right-xshift .value_required:n = true ,
8625   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8626   xshift .value_required:n = true ,
8627   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8628   delimiters / color .value_required:n = true ,
8629   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8630   slim .default:n = true ,
8631   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8632   hlines .default:n = all ,
8633   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8634   vlines .default:n = all ,
8635   hvlines .meta:n = { hlines, vlines } ,
8636   hvlines .value_forbidden:n = true
8637 }
8638 \keys_define:nn { nicematrix }
8639 {
8640   SubMatrix .inherit:n = nicematrix / sub-matrix ,
8641   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8642   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8643   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8644 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8645 \keys_define:nn { nicematrix / SubMatrix }
8646 {
8647   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8648   delimiters / color .value_required:n = true ,
8649   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8650   hlines .default:n = all ,
8651   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8652   vlines .default:n = all ,
8653   hvlines .meta:n = { hlines, vlines } ,
8654   hvlines .value_forbidden:n = true ,
8655   name .code:n =
8656     \tl_if_empty:nTF { #1 }
8657     { \@@_error:n { Invalid-name } }
8658     {
8659       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8660       {
8661         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8662         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8663         {
8664           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8665           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8666         }
8667       }
8668       { \@@_error:n { Invalid-name } }
8669     } ,
8670   name .value_required:n = true ,
8671   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8672   rules .value_required:n = true ,
8673   code .tl_set:N = \l_@@_code_tl ,
8674   code .value_required:n = true ,
8675   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
8676 }

```



```

8677 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
8678 {
8679   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8680   {
8681     \SubMatrix { #1 } { #2 } { #3 } { #4 }
8682     [
8683       delimiters / color = \l_@@_delimiters_color_tl ,
8684       hlines = \l_@@_submatrix_hlines_clist ,
8685       vlines = \l_@@_submatrix_vlines_clist ,
8686       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8687       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8688       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8689       slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8690       #5
8691     ]
8692   }
8693   \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8694   \ignorespaces
8695 }
8696 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8697 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8698 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8699 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8700 {
8701   \seq_gput_right:Ne \g_@@_submatrix_seq
8702   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8703   { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8704   { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8705   { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8706   { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8707 }
8708 }

```

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).

```

8709 \NewDocumentCommand \@@_compute_i_j:nn
8710 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8711 { \@@_compute_i_j:nnnn #1 #2 }
8712 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8713 {
8714   \def \l_@@_first_i_tl { #1 }
8715   \def \l_@@_first_j_tl { #2 }
8716   \def \l_@@_last_i_tl { #3 }
8717   \def \l_@@_last_j_tl { #4 }
8718   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8719     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8720   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8721     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8722   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8723     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8724   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8725     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8726 }

```

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`.

```

8727 \hook_gput_code:nnn { begindocument } { . }
8728 {
8729   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
8730   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
8731     { \@@_sub_matrix:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
8732 }
8733 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
8734 {
8735   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8736   \@@_compute_i_j:nn { #2 } { #3 }
8737   \int_compare:nNnT { \l_@@_first_i_tl } = { \l_@@_last_i_tl }
8738   { \def \arraystretch { 1 } }
8739   \bool_lazy_or:nnTF
8740     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
8741     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
8742     { \@@_error:nn { Construct~too-large } { \SubMatrix } }
8743   {
8744     \str_clear_new:N \l_@@_submatrix_name_str
8745     \keys_set:nn { nicematrix / SubMatrix } { #5 }
8746     \pgfpicture
8747     \pgfrememberpicturepositiononpagetrue
8748     \pgf@relevantforpicturesizefalse
8749     \pgfset { inner~sep = \c_zero_dim }
8750     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8751     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curriification.

```

8752   \bool_if:NTF \l_@@_submatrix_slim_bool
8753   { \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl } }
8754   { \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int } }
8755   {
8756     \cs_if_exist:cT
8757     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8758     {
8759       \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8760       \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
8761       { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8762     }
8763     \cs_if_exist:cT
8764     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8765     {
8766       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8767       \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
8768       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8769     }
8770   }
8771   \dim_compare:nNnTF { \l_@@_x_initial_dim } = { \c_max_dim }
8772   { \@@_error:nn { Impossible~delimiter } { left } }
8773   {
8774     \dim_compare:nNnTF { \l_@@_x_final_dim } = { - \c_max_dim }

```

```

8775         { \@@_error:nn { Impossible-delimiter } { right } }
8776         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8777     }
8778     \endpgfpicture
8779 }
8780 \group_end:
8781 \ignorespaces
8782 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8783 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8784 {
8785     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8786     \dim_set:Nn \l_@@_y_initial_dim
8787     {
8788         \fp_to_dim:n
8789         {
8790             \pgf@y
8791             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8792         }
8793     }
8794     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8795     \dim_set:Nn \l_@@_y_final_dim
8796     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8797     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
8798     {
8799         \cs_if_exist:cT
8800         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8801         {
8802             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8803             \dim_set:Nn \l_@@_y_initial_dim
8804             { \dim_max:nn { \l_@@_y_initial_dim } { \pgf@y } }
8805         }
8806         \cs_if_exist:cT
8807         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8808         {
8809             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8810             \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
8811             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8812         }
8813     }
8814     \dim_set:Nn \l_tmpa_dim
8815     {
8816         \l_@@_y_initial_dim - \l_@@_y_final_dim +
8817         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8818     }
8819     \dim_zero:N \nullldelimiterspace

```

We will draw the rules in the \SubMatrix.

```

8820     \group_begin:
8821     \pgfsetlinewidth { 1.1 \arrayrulewidth }
8822     \@@_set_CTarc:o \l_@@_rules_color_tl
8823     \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`.

```

8824     \seq_map_inline:Nn \g_@@_cols_vlism_seq
8825     {
8826         \int_compare:nNnT { \l_@@_first_j_tl } < { ##1 }
8827         {
8828             \int_compare:nNnT
8829             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }

```

```

8830         {
First, we extract the value of the abscissa of the rule we have to draw.
8831         \@_qpoint:n { col - ##1 }
8832         \pgfpathmoveto { \pgfpoint \pgf@x \l_@_y_initial_dim }
8833         \pgfpathlineto { \pgfpoint \pgf@x \l_@_y_final_dim }
8834         \pgfusepathqstroke
8835     }
8836 }
8837 }

```

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.

```

8838 \str_if_eq:eeTF \l_@_submatrix_vlines_clist { all }
8839 { \int_step_inline:nn { \l_@_last_j_tl - \l_@_first_j_tl } }
8840 { \clist_map_inline:Nn \l_@_submatrix_vlines_clist }
8841 {
8842     \bool_lazy_and:nnTF
8843     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
8844     {
8845         \int_compare_p:nNn
8846         { ##1 } < { \l_@_last_j_tl - \l_@_first_j_tl + 1 } }
8847     {
8848         \@_qpoint:n { col - \int_eval:n { ##1 + \l_@_first_j_tl } }
8849         \pgfpathmoveto { \pgfpoint \pgf@x \l_@_y_initial_dim }
8850         \pgfpathlineto { \pgfpoint \pgf@x \l_@_y_final_dim }
8851         \pgfusepathqstroke
8852     }
8853     { \@_error:nmn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8854 }

```

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.

```

8855 \str_if_eq:eeTF \l_@_submatrix_hlines_clist { all }
8856 { \int_step_inline:nn { \l_@_last_i_tl - \l_@_first_i_tl } }
8857 { \clist_map_inline:Nn \l_@_submatrix_hlines_clist }
8858 {
8859     \bool_lazy_and:nnTF
8860     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
8861     {
8862         \int_compare_p:nNn
8863         { ##1 } < { \l_@_last_i_tl - \l_@_first_i_tl + 1 } }
8864     {
8865         \@_qpoint:n { row - \int_eval:n { ##1 + \l_@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8866 \group_begin:
We compute in \l_tmpa_dim the  $x$ -value of the left end of the rule.
8867 \dim_set:Nn \l_tmpa_dim
8868 { \l_@_x_initial_dim - \l_@_submatrix_left_xshift_dim }
8869 \str_case:nn { #1 }
8870 {
8871     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } } }
8872     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } } }
8873     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } } }
8874 }
8875 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8876 \dim_set:Nn \l_tmpb_dim
8877 { \l_@_x_final_dim + \l_@_submatrix_right_xshift_dim }
8878 \str_case:nn { #2 }
8879 {
8880     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }

```

```

8881         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8882         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8883     }
8884     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8885     \pgfusepathqstroke
8886     \group_end:
8887 }
8888 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8889 }

```

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).

```

8890 \str_if_empty:NF \l_@@_submatrix_name_str
8891 {
8892     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8893     \l_@@_x_initial_dim \l_@@_y_initial_dim
8894     \l_@@_x_final_dim \l_@@_y_final_dim
8895 }
8896 \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`.

```

8897 \begin { pgfscope }
8898 \pgftransformshift
8899 {
8900     \pgfpoint
8901     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8902     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8903 }
8904 \str_if_empty:NTF \l_@@_submatrix_name_str
8905 { \@@_node_left:nn #1 { } }
8906 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8907 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8908 \pgftransformshift
8909 {
8910     \pgfpoint
8911     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8912     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8913 }
8914 \str_if_empty:NTF \l_@@_submatrix_name_str
8915 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8916 {
8917     \@@_node_right:nnnn #2
8918     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8919 }

```

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`.

```

8920 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8921 \flag_clear_new:N \l_@@_code_flag
8922 \l_@@_code_tl
8923 }

```

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`.

```

8924 \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.

```
8925 \cs_new:Npn \@@_pgfpointanchor:n #1
8926 { \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`.

```
8927 \cs_new:Npn \@@_pgfpointanchor_i:n #1
8928 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }

8929 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
8930 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
8931 \str_if_empty:nTF { #1 }
```

First, when the name of the name begins with `\tikz@pp@name`.

```
8932 { \@@_pgfpointanchor_iv:w #2 }
```

And now, when there is no `\tikz@pp@name`.

```
8933 { \@@_pgfpointanchor_ii:n { #1 } }
8934 }
```

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`).

```
8935 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
8936 { \@@_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`.

```
8937 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

8938 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
8939 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
8940 \str_if_empty:nTF { #2 }
```

First the case where the argument does *not* contain an hyphen.

```
8941 { \@@_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`).

```
8942 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8943 }
```

The following function is for the case when the name contains an hyphen.

```
8944 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8945 {
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
8946 \@@_env:
8947 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8948 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8949 }
```

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.

```

8950 \tl_const:Nn \c_@@_integers_alist_tl
8951 {
8952   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8953   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8954   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8955   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8956 }

8957 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
8958 {

```

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`.

```

8959   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8960   {
8961     \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

8962   \@@_env: -
8963   \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8964     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8965     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8966   }
8967   {
8968     \str_if_eq:eeTF { #1 } { last }
8969     {
8970       \flag_raise:N \l_@@_code_flag
8971       \@@_env: -
8972       \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8973         { \int_eval:n { \l_@@_last_i_tl + 1 } }
8974         { \int_eval:n { \l_@@_last_j_tl + 1 } }
8975     }
8976     { #1 }
8977   }
8978 }

```

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`).

```

8979 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8980 {
8981   \pgfnode
8982     { rectangle }
8983     { east }
8984     {
8985       \nullfont
8986       \c_math_toggle_token
8987       \@@_color:o \l_@@_delimiters_color_tl
8988       \left #1
8989       \vcenter
8990       {
8991         \nullfont
8992         \hrule \@height \l_tmpa_dim
8993         \@depth \c_zero_dim

```

```

8994         \@width \c_zero_dim
8995     }
8996     \right .
8997     \c_math_toggle_token
8998 }
8999 { #2 }
9000 { }
9001 }

```

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.

```

9002 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9003 {
9004     \pgfnode
9005     { rectangle }
9006     { west }
9007     {
9008         \nullfont
9009         \c_math_toggle_token
9010         \colorlet { current-color } { . }
9011         \@@_color:o \l_@@_delimiters_color_tl
9012         \left .
9013         \vcenter
9014         {
9015             \nullfont
9016             \hrule \@height \l_tmpa_dim
9017                 \@depth \c_zero_dim
9018                 \@width \c_zero_dim
9019         }
9020         \right #1
9021         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9022         ^ { \color { current-color } \smash { #4 } }
9023         \c_math_toggle_token
9024     }
9025     { #2 }
9026     { }
9027 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9028 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9029 {
9030     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9031     \ignorespaces
9032 }
9033 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9034 {
9035     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9036     \ignorespaces
9037 }
9038 \keys_define:nn { nicematrix / Brace }
9039 {
9040     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9041     left-shorten .default:n = true ,
9042     left-shorten .value_forbidden:n = true ,

```



```

9043   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9044   right-shorten .default:n = true ,
9045   right-shorten .value_forbidden:n = true ,
9046   shorten .meta:n = { left-shorten , right-shorten } ,
9047   shorten .value_forbidden:n = true ,
9048   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9049   yshift .value_required:n = true ,
9050   yshift .initial:n = \c_zero_dim ,
9051   color .tl_set:N = \l_tmpa_tl ,
9052   color .value_required:n = true ,
9053   unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
9054 }

```

#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.

```

9055 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9056 {
9057   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9058   \@@_compute_i_j:nn { #1 } { #2 }
9059   \bool_lazy_or:nnTF
9060     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9061     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9062   {
9063     \str_if_eq:eeTF { #5 } { under }
9064     { \@@_error:nn { Construct~too~large } { \UnderBrace } }
9065     { \@@_error:nn { Construct~too~large } { \OverBrace } }
9066   }
9067   {
9068     \tl_clear:N \l_tmpa_tl
9069     \keys_set:nn { nicematrix / Brace } { #4 }
9070     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9071     \pgfpicture
9072     \pgfrememberpicturepositiononpagetrue
9073     \pgf@relevantforpicturesizefalse
9074     \bool_if:NT \l_@@_brace_left_shorten_bool
9075     {
9076       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9077       \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9078       {
9079         \cs_if_exist:cT
9080           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9081         {
9082           \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9083
9084           \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9085             { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9086         }
9087       }
9088     }
9089     \bool_lazy_or:nnT
9090       { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9091       { \dim_compare_p:nNn { \l_@@_x_initial_dim } = { \c_max_dim } }
9092     {
9093       \@@_qpoint:n { col - \l_@@_first_j_tl }
9094       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9095     }
9096     \bool_if:NT \l_@@_brace_right_shorten_bool
9097     {
9098       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9099       \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9100       {

```

```

9101         \cs_if_exist:cT
9102         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9103         {
9104             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9105             \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
9106             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9107         }
9108     }
9109 }
9110 \bool_lazy_or:nnT
9111 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9112 { \dim_compare_p:nNn { \l_@@_x_final_dim } = { - \c_max_dim } }
9113 {
9114     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9115     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9116 }
9117 \pgfset { inner~sep = \c_zero_dim }
9118 \str_if_eq:eeTF { #5 } { under }
9119 { \@@_underbrace_i:n { #3 } }
9120 { \@@_overbrace_i:n { #3 } }
9121 \endpgfpicture
9122 }
9123 \group_end:
9124 }

```

The argument is the text to put above the brace.

```

9125 \cs_new_protected:Npn \@@_overbrace_i:n #1
9126 {
9127     \@@_qpoint:n { row - \l_@@_first_i_tl }
9128     \pgftransformshift
9129     {
9130         \pgfpoint
9131         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9132         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9133     }
9134     \pgfnode
9135     { rectangle }
9136     { south }
9137     {
9138         \vtop
9139         {
9140             \group_begin:
9141             \everycr { }
9142             \halign
9143             {
9144                 \hfil ## \hfil \crcr
9145                 \bool_if:NTF \l_@@_tabular_bool
9146                 { \begin { tabular } { c } #1 \end { tabular } }
9147                 { $ \begin { array } { c } #1 \end { array } $ }
9148                 \cr
9149                 \c_math_toggle_token
9150                 \overbrace
9151                 {
9152                     \hbox_to_wd:nn
9153                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9154                     { }
9155                 }
9156                 \c_math_toggle_token
9157                 \cr
9158             }
9159             \group_end:
9160         }
9161     }
9162 }

```

```

9163     { }
9164 }

```

The argument is the text to put under the brace.

```

9165 \cs_new_protected:Npn \@@_underbrace_i:n #1
9166 {
9167   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9168   \pgftransformshift
9169   {
9170     \pgfpoint
9171     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9172     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9173   }
9174   \pgfnode
9175   { rectangle }
9176   { north }
9177   {
9178     \group_begin:
9179     \everycr { }
9180     \vbox
9181     {
9182       \halign
9183       {
9184         \hfil ## \hfil \crcr
9185         \c_math_toggle_token
9186         \underbrace
9187         {
9188           \hbox_to_wd:nn
9189           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9190           { }
9191         }
9192         \c_math_toggle_token
9193         \cr
9194         \bool_if:NTF \l_@@_tabular_bool
9195         { \begin { tabular } { c } #1 \end { tabular } }
9196         { $ \begin { array } { c } #1 \end { array } $ }
9197         \cr
9198       }
9199     }
9200     \group_end:
9201   }
9202   { }
9203   { }
9204 }

```

35 The commands HBrace et VBrace

```

9205 \hook_gput_code:nnn { begindocument } { . }
9206 {
9207   \cs_if_exist:cT { tikz@library@decorations.pathreplacing@loaded }
9208   {
9209     \tikzset
9210     {
9211       nicematrix / brace / .style =
9212       {
9213         decoration = { brace , raise = -0.15 em } ,
9214         decorate ,
9215       } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9216         nicematrix / mirrored-brace / .style =
9217         {
9218             nicematrix / brace ,
9219             decoration = mirror ,
9220         }
9221     }
9222 }
9223 }
```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9224 \keys_define:nn { nicematrix / Hbrace }
9225 {
9226     color .code:n = ,
9227     horizontal-label .code:n = ,
9228     horizontal-labels .code:n = ,
9229     shorten .code:n = ,
9230     shorten-start .code:n = ,
9231     shorten-end .code:n = ,
9232     unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9233 }
```

Here we need an “fully expandable” command.

```

9234 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9235 {
9236     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9237     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9238     { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9239 }
```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

9240 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9241 {
9242     \int_compare:nNnTF { \c@iRow } < { 2 }
9243     {
```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9244     \str_if_eq:nnTF { #2 } { * }
9245     {
9246         \NiceMatrixOptions { nullify-dots }
9247         \Ldots
9248         [
9249             line-style = nicematrix / brace ,
9250             #1 ,
9251             up =
9252             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9253         ]
9254     }
9255     {
9256         \Hdotsfor
9257         [
9258             line-style = nicematrix / brace ,
9259             #1 ,
9260             up =
9261             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9262         ]
9263         { #2 }
9264     }
9265 }
9266 {
9267     \str_if_eq:nnTF { #2 } { * }
```

```

9268 {
9269   \NiceMatrixOptions { nullify-dots }
9270   \Ldots
9271   [
9272     line-style = nicematrix / mirrored-brace ,
9273     #1 ,
9274     down =
9275       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9276   ]
9277 }
9278 {
9279   \Hdotsfor
9280   [
9281     line-style = nicematrix / mirrored-brace ,
9282     #1 ,
9283     down =
9284       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9285   ]
9286   { #2 }
9287 }
9288 }
9289 \keys_set:nn { nicematrix / Hbrace } { #1 }
9290 }

9291 \NewDocumentCommand { \@@_Vbrace } { 0 { } m m }
9292 {
9293   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9294   { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9295   { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9296 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

9297 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
9298 {
9299   \int_compare:nNnTF { \c@jCol } < { 2 }
9300   {
9301     \str_if_eq:nnTF { #2 } { * }
9302     {
9303       \NiceMatrixOptions { nullify-dots }
9304       \Vdots
9305       [
9306         Vbrace ,
9307         line-style = nicematrix / mirrored-brace ,
9308         #1 ,
9309         down =
9310           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9311       ]
9312     }
9313     {
9314       \Vdotsfor
9315       [
9316         Vbrace ,
9317         line-style = nicematrix / mirrored-brace ,
9318         #1 ,
9319         down =
9320           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9321       ]
9322       { #2 }
9323     }
9324   }
9325   {
9326     \str_if_eq:nnTF { #2 } { * }
9327     {

```

```

9328         \NiceMatrixOptions { nullify-dots }
9329         \Vdots
9330         [
9331             Vbrace ,
9332             line-style = nicematrix / brace ,
9333             #1 ,
9334             up =
9335             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9336         ]
9337     }
9338     {
9339         \Vdotsfor
9340         [
9341             Vbrace ,
9342             line-style = nicematrix / brace ,
9343             #1 ,
9344             up =
9345             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9346         ]
9347         { #2 }
9348     }
9349 }
9350 \keys_set:nn { nicematrix / Hbrace } { #1 }
9351 }

```

36 The command TikzEveryCell

```

9352 \bool_new:N \l_@@_not_empty_bool
9353 \bool_new:N \l_@@_empty_bool
9354
9355 \keys_define:nn { nicematrix / TikzEveryCell }
9356 {
9357     not-empty .code:n =
9358         \bool_lazy_or:nnTF
9359         { \l_@@_in_code_after_bool }
9360         { \g_@@_create_cell_nodes_bool }
9361         { \bool_set_true:N \l_@@_not_empty_bool }
9362         { \@@_error:n { detection-of-empty-cells } } ,
9363     not-empty .value_forbidden:n = true ,
9364     empty .code:n =
9365         \bool_lazy_or:nnTF
9366         { \l_@@_in_code_after_bool }
9367         { \g_@@_create_cell_nodes_bool }
9368         { \bool_set_true:N \l_@@_empty_bool }
9369         { \@@_error:n { detection-of-empty-cells } } ,
9370     empty .value_forbidden:n = true ,
9371     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
9372 }
9373
9374
9375 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9376 {
9377     \IfPackageLoadedTF { tikz }
9378     {
9379         \group_begin:
9380         \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.

```

9381         \tl_set:Nn \l_tmpa_tl { { #2 } }
9382         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9383         { \@@_for_a_block:nnnnn ##1 }

```

```

9384 \@@_all_the_cells:
9385 \group_end:
9386 }
9387 { \@@_error:n { TikzEveryCell~without~tikz } }
9388 }
9389
9390 \tl_new:N \l_@@_i_tl
9391 \tl_new:N \l_@@_j_tl
9392
9393
9394 \cs_new_protected:Nn \@@_all_the_cells:
9395 {
9396   \int_step_inline:nn \c@iRow
9397   {
9398     \int_step_inline:nn \c@jCol
9399     {
9400       \cs_if_exist:cF { cell - ##1 - #####1 }
9401       {
9402         \clist_if_in:NcF \l_@@_corners_cells_clist
9403         { ##1 - #####1 }
9404         {
9405           \bool_set_false:N \l_tmpa_bool
9406           \cs_if_exist:cTF
9407           { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
9408           {
9409             \bool_if:NF \l_@@_empty_bool
9410             { \bool_set_true:N \l_tmpa_bool }
9411           }
9412           {
9413             \bool_if:NF \l_@@_not_empty_bool
9414             { \bool_set_true:N \l_tmpa_bool }
9415           }
9416           \bool_if:NT \l_tmpa_bool
9417           {
9418             \@@_block_tikz:nnnnn
9419             \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
9420           }
9421         }
9422       }
9423     }
9424   }
9425 }
9426
9427 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9428 {
9429   \bool_if:NF \l_@@_empty_bool
9430   {
9431     \@@_block_tikz:nnnnn
9432     \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9433   }
9434   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9435 }
9436
9437 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9438 {
9439   \int_step_inline:nnn { #1 } { #3 }
9440   {
9441     \int_step_inline:nnn { #2 } { #4 }
9442     { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9443   }
9444 }

```

37 The command \ShowCellNames

```

9445 \NewDocumentCommand \@@_ShowCellNames { }
9446 {
9447   \bool_if:NT \l_@@_in_code_after_bool
9448   {
9449     \pgfpicture
9450     \pgfrememberpicturepositiononpagetrue
9451     \pgf@relevantforpicturesizefalse
9452     \pgfpathrectanglecorners
9453     { \@@_qpoint:n { 1 } }
9454     {
9455       \@@_qpoint:n
9456       { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
9457     }
9458     \pgfsetfillopacity { 0.75 }
9459     \pgfsetfillcolor { white }
9460     \pgfusepathqfill
9461     \endpgfpicture
9462   }
9463   \dim_gzero_new:N \g_@@_tmpc_dim
9464   \dim_gzero_new:N \g_@@_tmpd_dim
9465   \dim_gzero_new:N \g_@@_tmpe_dim
9466   \int_step_inline:nn { \c@iRow }
9467   {
9468     \bool_if:NTF \l_@@_in_code_after_bool
9469     {
9470       \pgfpicture
9471       \pgfrememberpicturepositiononpagetrue
9472       \pgf@relevantforpicturesizefalse
9473     }
9474     { \begin { pgfpicture } }
9475     \@@_qpoint:n { row - ##1 }
9476     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9477     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9478     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9479     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9480     \bool_if:NTF \l_@@_in_code_after_bool
9481     { \endpgfpicture }
9482     { \end { pgfpicture } }
9483     \int_step_inline:nn { \c@jCol }
9484     {
9485       \hbox_set:Nn \l_tmpa_box
9486       {
9487         \normalfont \Large \sffamily \bfseries
9488         \bool_if:NTF \l_@@_in_code_after_bool
9489         { \color { red } }
9490         { \color { red ! 50 } }
9491         ##1 - ####1
9492       }
9493       \bool_if:NTF \l_@@_in_code_after_bool
9494       {
9495         \pgfpicture
9496         \pgfrememberpicturepositiononpagetrue
9497         \pgf@relevantforpicturesizefalse
9498       }
9499       { \begin { pgfpicture } }
9500       \@@_qpoint:n { col - ####1 }
9501       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9502       \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9503       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9504       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x

```



```

9505 \bool_if:NTF \l_@@_in_code_after_bool
9506 { \endpgfpicture }
9507 { \end { pgfpicture } }
9508 \fp_set:Nn \l_tmpa_fp
9509 {
9510   \fp_min:nn
9511   {
9512     \fp_min:nn
9513     { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9514     { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9515   }
9516   { 1.0 }
9517 }
9518 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9519 \pgfpicture
9520 \pgfrememberpicturerepositiononpagetrue
9521 \pgf@relevantforpicturesizefalse
9522 \pgftransformshift
9523 {
9524   \pgfpoint
9525   { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9526   { \dim_use:N \g_tmpa_dim }
9527 }
9528 \pgfnode
9529 { rectangle }
9530 { center }
9531 { \box_use:N \l_tmpa_box }
9532 { }
9533 { }
9534 \endpgfpicture
9535 }
9536 }
9537 }

```

38 We process the options at package loading

We process the options when the package is loaded (with `\usepackage`) but we recommend to use `\NiceMatrixOptions` instead.

We must process these options after the definition of the environment `{NiceMatrix}` because the option `renew-matrix` executes the code `\cs_set_eq:NN \env@matrix \NiceMatrix`.

Of course, the command `\NiceMatrix` must be defined before such an instruction is executed.

The boolean `\g_@@_footnotehyper_bool` will indicate if the option `footnotehyper` is used.

```

9538 \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.

```

9539 \bool_new:N \g_@@_footnote_bool
9540 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9541 {
9542   You-have-used-the-key~' \l_keys_key_str '~when-loading-nicematrix~
9543   but-that-key-is-unknown. \\
9544   It-will-be-ignored. \\
9545   For-a-list-of-the-available-keys,~type-H~<return>.
9546 }
9547 {
9548   The-available-keys-are~(in-alphabetic-order):~
9549   footnote,~
9550   footnotehyper,~
9551   messages-for-Overleaf,~
9552   renew-dots-and~

```

```

9553     renew-matrix.
9554 }
9555 \keys_define:nn { nicematrix }
9556 {
9557     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9558     renew-dots .value_forbidden:n = true ,
9559     renew-matrix .code:n = \@@_renew_matrix: ,
9560     renew-matrix .value_forbidden:n = true ,
9561     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9562     footnote .bool_set:N = \g_@@_footnote_bool ,
9563     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9564     unknown .code:n = \@@_error:n { Unknown~key~for~package }
9565 }
9566 \ProcessKeyOptions

9567 \@@_msg_new:nn { footnote-with-footnotehyper-package }
9568 {
9569     You~can't~use~the~option~'footnote'~because~the~package~
9570     footnotehyper~has~already~been~loaded.~
9571     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9572     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9573     of~the~package~footnotehyper.\\
9574     The~package~footnote~won't~be~loaded.
9575 }

9576 \@@_msg_new:nn { footnotehyper-with-footnote-package }
9577 {
9578     You~can't~use~the~option~'footnotehyper'~because~the~package~
9579     footnote~has~already~been~loaded.~
9580     If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9581     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9582     of~the~package~footnote.\\
9583     The~package~footnotehyper~won't~be~loaded.
9584 }

9585 \bool_if:NT \g_@@_footnote_bool
9586 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9587     \IfClassLoadedTF { beamer }
9588     { \bool_set_false:N \g_@@_footnote_bool }
9589     {
9590         \IfPackageLoadedTF { footnotehyper }
9591         { \@@_error:n { footnote-with-footnotehyper~package } }
9592         { \usepackage { footnote } }
9593     }
9594 }

9595 \bool_if:NT \g_@@_footnotehyper_bool
9596 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9597     \IfClassLoadedTF { beamer }
9598     { \bool_set_false:N \g_@@_footnote_bool }
9599     {
9600         \IfPackageLoadedTF { footnote }
9601         { \@@_error:n { footnotehyper-with-footnote~package } }
9602         { \usepackage { footnotehyper } }
9603     }
9604     \bool_set_true:N \g_@@_footnote_bool
9605 }

```

The flag `\g_@@_footnote_bool` is raised and so, we will only have to test `\g_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

39 About the package underscore

If the user loads the package `underscore`, it must be loaded *before* the package `nicematrix`. If it is loaded after, we raise an error.

```

9606 \bool_new:N \l_@@_underscore_loaded_bool
9607 \IfPackageLoadedT { underscore }
9608 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9609 \hook_gput_code:nnn { begindocument } { . }
9610 {
9611   \bool_if:NF \l_@@_underscore_loaded_bool
9612   {
9613     \IfPackageLoadedT { underscore }
9614     { \@@_error:n { underscore-after-nicematrix } }
9615   }
9616 }

```

40 Error messages of the package

```

9617 \str_const:Ne \c_@@_available_keys_str
9618 {
9619   \bool_if:nTF { ! \g_@@_messages_for_Overleaf_bool }
9620   { For~a~list~of~the~available~keys,~type~H~<return>. }
9621   { }
9622 }
9623 \seq_new:N \g_@@_types_of_matrix_seq
9624 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9625 {
9626   NiceMatrix ,
9627   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9628 }
9629 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9630 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_error_too_much_cols:` is triggered. This command raises an error but also tries to give the best information to the user in the error message. The command `\seq_if_in:NoF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

9631 \cs_new_protected:Npn \@@_error_too_much_cols:
9632 {
9633   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9634   { \@@_fatal:nn { too-much-cols-for-array } }
9635   \int_compare:nNnT { \l_@@_last_col_int } = { -2 }
9636   { \@@_fatal:n { too-much-cols-for-matrix } }
9637   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
9638   { \@@_fatal:n { too-much-cols-for-matrix } }
9639   \bool_if:NF \l_@@_last_col_without_value_bool
9640   { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9641 }

```

The following command must *not* be protected since it's used in an error message.

```

9642 \cs_new:Npn \@@_message_hdotsfor:
9643 {
9644   \tl_if_empty:oF \g_@@_HVDotsfor_lines_tl
9645   { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
9646     \token_to_str:N \Hbrace \ is~incorrect. }
9647 }

9648 \cs_new_protected:Npn \@@_Hline_in_cell:
9649 { \@@_fatal:n { Misuse~of~Hline } }

9650 \@@_msg_new:nn { Misuse~of~Hline }
9651 {
9652   Misuse~of~Hline. \\
9653   \token_to_str:N \Hline\ must~be~used~only~at~the~beginning~of~a~row.\\
9654   That~error~is~fatal.
9655 }

9656 \@@_msg_new:nn { hvlines,~rounded~corners~and~corners }
9657 {
9658   Incompatible~options.\\
9659   You~should~not~use~'hvlines',~'rounded~corners'~and~'corners'~at~the~same~time.\\
9660   The~output~will~not~be~reliable.
9661 }

9662 \@@_msg_new:nn { key~color~inside }
9663 {
9664   Key~deprecated.\\
9665   The~key~'color~inside'~(and~its~alias~'colortbl~like')~is~now~point~less~
9666   and~have~been~deprecated.\\
9667   You~won't~have~similar~message~till~the~end~of~the~document.
9668 }

9669 \@@_msg_new:nn { invalid~weight }
9670 {
9671   Unknown~key.\\
9672   The~key~' \l_keys_key_str ' ~of~your~column~X~is~unknown~and~will~be~ignored.
9673 }

9674 \@@_msg_new:nn { last~col~not~used }
9675 {
9676   Column~not~used.\\
9677   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
9678   in~your~\@@_full_name_env: .~
9679   However,~you~can~go~on.
9680 }

9681 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
9682 {
9683   Too~much~columns.\\
9684   In~the~row~ \int_eval:n { \c@iRow },~
9685   you~try~to~use~more~columns~
9686   than~allowed~by~your~ \@@_full_name_env: .
9687   \@@_message_hdotsfor: \
9688   The~maximal~number~of~columns~is~ \int_eval:n { \l_@@_last_col_int - 1 }~
9689   (plus~the~exterior~columns).~This~error~is~fatal.
9690 }

9691 \@@_msg_new:nn { too~much~cols~for~matrix }
9692 {
9693   Too~much~columns.\\
9694   In~the~row~ \int_eval:n { \c@iRow },~
9695   you~try~to~use~more~columns~than~allowed~by~your~ \@@_full_name_env: .
9696   \@@_message_hdotsfor: \
9697   Recall~that~the~maximal~number~of~columns~for~a~matrix~
9698   (excepted~the~potential~exterior~columns)~is~fixed~by~the~
9699   LaTeX~counter~'MaxMatrixCols'.~
9700   Its~current~value~is~ \int_use:N \c@MaxMatrixCols \
9701   (use~ \token_to_str:N \setcounter \ to~change~that~value).~

```

```

9702     This-error-is-fatal.
9703 }

9704 \@@_msg_new:nn { too-much-cols-for-array }
9705 {
9706     Too-much-columns.\\
9707     In-the-row \int_eval:n { \c@iRow } ,~
9708     ~you-try-to-use-more-columns-than-allowed-by-your~
9709     \@@_full_name_env: . \@@_message_hdotsfor: \ The-maximal-number-of-columns-is~
9710     \int_use:N \g_@@_static_num_of_col_int \
9711     \bool_if:nT
9712     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
9713     { ~(plus-the-exterior-ones) }
9714     since-the-preamble-is~' \g_@@_user_preamble_tl '.\\
9715     This-error-is-fatal.
9716 }

9717 \@@_msg_new:nn { columns-not-used }
9718 {
9719     Columns-not-used.\\
9720     The-preamble-of-your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '~
9721     It-announces~ \int_use:N \g_@@_static_num_of_col_int \
9722     columns-but-you-only-used~ \int_use:N \c@jCol .\\
9723     The-columns-you-did-not-used-won't-be-created.\\
9724     You-won't-have-similar-warning-till-the-end-of-the-document.
9725 }

9726 \@@_msg_new:nn { empty-preamble }
9727 {
9728     Empty-preamble.\\
9729     The-preamble-of-your~ \@@_full_name_env: \ is-empty.\\
9730     This-error-is-fatal.
9731 }

9732 \@@_msg_new:nn { in-first-col }
9733 {
9734     Erroneous-use.\\
9735     You-can't-use-the-command~#1 in-the-first-column~(number~0)~of-the-array.\\
9736     That-command-will-be-ignored.
9737 }

9738 \@@_msg_new:nn { in-last-col }
9739 {
9740     Erroneous-use.\\
9741     You-can't-use-the-command~#1 in-the-last-column~(exterior)~of-the-array.\\
9742     That-command-will-be-ignored.
9743 }

9744 \@@_msg_new:nn { in-first-row }
9745 {
9746     Erroneous-use.\\
9747     You-can't-use-the-command~#1 in-the-first-row~(number~0)~of-the-array.\\
9748     That-command-will-be-ignored.
9749 }

9750 \@@_msg_new:nn { in-last-row }
9751 {
9752     Erroneous-use.\\
9753     You-can't-use-the-command~#1 in-the-last-row~(exterior)~of-the-array.\\
9754     That-command-will-be-ignored.
9755 }

9756 \@@_msg_new:nn { TopRule-without-booktabs }
9757 {
9758     Erroneous-use.\\
9759     You-can't-use-the-command~ #1 because~'booktabs'~is-not-loaded.\\
9760     That-command-will-be-ignored.
9761 }

```

```

9762 \@@_msg_new:nn { TopRule~without~tikz }
9763 {
9764   Erroneous~use.\\
9765   You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
9766   That~command~will~be~ignored.
9767 }
9768 \@@_msg_new:nn { caption~outside~float }
9769 {
9770   Key~caption~forbidden.\\
9771   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9772   environment~(such~as~\{table\}).~This~key~will~be~ignored.
9773 }
9774 \@@_msg_new:nn { short~caption~without~caption }
9775 {
9776   You~should~not~use~the~key~'short~caption'~without~'caption'.~
9777   However,~your~'short~caption'~will~be~used~as~'caption'.
9778 }
9779 \@@_msg_new:nn { double~closing~delimiter }
9780 {
9781   Double~delimiter.\\
9782   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9783   delimiter.~This~delimiter~will~be~ignored.
9784 }
9785 \@@_msg_new:nn { delimiter~after~opening }
9786 {
9787   Double~delimiter.\\
9788   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9789   delimiter.~That~delimiter~will~be~ignored.
9790 }
9791 \@@_msg_new:nn { bad~option~for~line~style }
9792 {
9793   Bad~line~style.\\
9794   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9795   is~'standard'.~That~key~will~be~ignored.
9796 }
9797 \@@_msg_new:nn { corners~with~no~cell~nodes }
9798 {
9799   Incompatible~keys.\\
9800   You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
9801   is~in~force.\\
9802   If~you~go~on,~that~key~will~be~ignored.
9803 }
9804 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
9805 {
9806   Incompatible~keys.\\
9807   You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
9808   is~in~force.\\
9809   If~you~go~on,~those~extra~nodes~won't~be~created.
9810 }
9811 \@@_msg_new:nn { Identical~notes~in~caption }
9812 {
9813   Identical~tabular~notes.\\
9814   You~can't~put~several~notes~with~the~same~content~in~
9815   \token_to_str:N \caption \ (but~you~can~in~the~main~tabular).\\
9816   If~you~go~on,~the~output~will~probably~be~erroneous.
9817 }
9818 \@@_msg_new:nn { tabularnote~below~the~tabular }
9819 {
9820   \token_to_str:N \tabularnote \ forbidden\\
9821   You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~

```

```

9822 of~your~tabular~because~the~caption~will~be~composed~below~
9823 the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9824 key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .\\
9825 Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
9826 no~similar~error~will~raised~in~this~document.
9827 }
9828 \@@_msg_new:nn { Unknown~key~for~rules }
9829 {
9830   Unknown~key.\\
9831   There~is~only~two~keys~available~here:~width~and~color.\\
9832   Your~key~' \l_keys_key_str '~will~be~ignored.
9833 }
9834 \@@_msg_new:nn { Unknown~key~for~Hbrace }
9835 {
9836   Unknown~key.\\
9837   You~have~used~the~key~' \l_keys_key_str '~but~the~only~
9838   keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
9839   and~ \token_to_str:N \Vbrace \ are:~'color',~
9840   'horizontal-label(s)',~'shorten'~'shorten-end'~
9841   and~'shorten-start'.\\
9842   That~error~is~fatal.
9843 }
9844 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9845 {
9846   Unknown~key.\\
9847   There~is~only~two~keys~available~here:~
9848   'empty'~and~'not-empty'.\\
9849   Your~key~' \l_keys_key_str '~will~be~ignored.
9850 }
9851 \@@_msg_new:nn { Unknown~key~for~rotate }
9852 {
9853   Unknown~key.\\
9854   The~only~key~available~here~is~'c'.\\
9855   Your~key~' \l_keys_key_str '~will~be~ignored.
9856 }
9857 \@@_msg_new:nnn { Unknown~key~for~custom-line }
9858 {
9859   Unknown~key.\\
9860   The~key~' \l_keys_key_str '~is~unknown~in~a~'custom-line'.~
9861   It~you~go~on,~you~will~probably~have~other~errors. \\
9862   \c_@@_available_keys_str
9863 }
9864 {
9865   The~available~keys~are~(in~alphabetic~order):~
9866   ccommand,~
9867   color,~
9868   command,~
9869   dotted,~
9870   letter,~
9871   multiplicity,~
9872   sep-color,~
9873   tikz,~and~total-width.
9874 }
9875 \@@_msg_new:nnn { Unknown~key~for~xdots }
9876 {
9877   Unknown~key.\\
9878   The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9879   \c_@@_available_keys_str
9880 }
9881 {
9882   The~available~keys~are~(in~alphabetic~order):~

```

```

9883     'color',~
9884     'horizontal(s)-labels',~
9885     'inter',~
9886     'line-style',~
9887     'radius',~
9888     'shorten',~
9889     'shorten-end'~and~'shorten-start'.
9890 }

9891 \@@_msg_new:nn { Unknown-key~for~rowcolors }
9892 {
9893     Unknown-key.\\
9894     As~for~now,~there-is~only~two-keys~available~here:~'cols'~and~'respect-blocks'~
9895     (and~you~try~to~use~' \l_keys_key_str ')\
9896     That~key~will~be~ignored.
9897 }

9898 \@@_msg_new:nn { label-without-caption }
9899 {
9900     You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
9901     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9902 }

9903 \@@_msg_new:nn { W-warning }
9904 {
9905     Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
9906     (row~ \int_use:N \c@iRow ).
9907 }

9908 \@@_msg_new:nn { Construct-too-large }
9909 {
9910     Construct-too-large.\\
9911     Your~command~ \token_to_str:N #1
9912     can't~be~drawn~because~your~matrix~is~too~small.\\
9913     That~command~will~be~ignored.
9914 }

9915 \@@_msg_new:nn { underscore~after~nicematrix }
9916 {
9917     Problem~with~'underscore'.\\
9918     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9919     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9920     ' \token_to_str:N \Cdots \token_to_str:N _
9921     \{ n \token_to_str:N \text \{ ~times \} \}' .
9922 }

9923 \@@_msg_new:nn { ampersand~in~light-syntax }
9924 {
9925     Ampersand~forbidden.\\
9926     You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
9927     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9928 }

9929 \@@_msg_new:nn { double-backslash~in~light-syntax }
9930 {
9931     Double~backslash~forbidden.\\
9932     You~can't~use~ \token_to_str:N \\
9933     ~to~separate~rows~because~the~key~'light-syntax'~
9934     is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
9935     (set~by~the~key~'end-of-row').~This~error~is~fatal.
9936 }

9937 \@@_msg_new:nn { hlines~with~color }
9938 {
9939     Incompatible~keys.\\
9940     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9941     \token_to_str:N \Block \ when~the~key~'color'~or~'draw'~is~used.\\
9942     However,~you~can~put~several~commands~ \token_to_str:N \Block.\\

```



```

9943     Your~key~will~be~discarded.
9944 }
9945 \@@_msg_new:nn { bad~value~for~baseline }
9946 {
9947     Bad~value~for~baseline.\\
9948     The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
9949     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9950     \int_use:N \g_@@_row_total_int \ or~equal~to~'t',~'c'~or~'b'~or~of~
9951     the~form~'line-i'.\\
9952     A~value~of~1~will~be~used.
9953 }
9954 \@@_msg_new:nn { detection~of~empty~cells }
9955 {
9956     Problem~with~'not~empty'\\
9957     For~technical~reasons,~you~must~activate~
9958     'create~cell~nodes'~in~ \token_to_str:N \createBefore \
9959     in~order~to~use~the~key~' \l_keys_key_str '.\\
9960     That~key~will~be~ignored.
9961 }
9962 \@@_msg_new:nn { siunitx~not~loaded }
9963 {
9964     siunitx~not~loaded\\
9965     You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9966     That~error~is~fatal.
9967 }
9968 \@@_msg_new:nn { Invalid~name }
9969 {
9970     Invalid~name.\\
9971     You~can't~give~the~name~' \l_keys_value_tl '~to~a~ \token_to_str:N
9972     \SubMatrix \ of~your~ \@@_full_name_env: .\\
9973     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9974     This~key~will~be~ignored.
9975 }
9976 \@@_msg_new:nn { Hbrace~not~allowed }
9977 {
9978     Command~not~allowed.\\
9979     You~can't~use~the~command~ \token_to_str:N #1
9980     because~you~have~not~loaded~
9981     \IfPackageLoadedTF { tikz }
9982     { the~TikZ~library~'decorations.pathreplacing'~.~Use~ }
9983     { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
9984     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\
9985     That~command~will~be~ignored.
9986 }
9987 \@@_msg_new:nn { Vbrace~not~allowed }
9988 {
9989     Command~not~allowed.\\
9990     You~can't~use~the~command~ \token_to_str:N \Vbrace \
9991     because~you~have~not~loaded~TikZ~
9992     and~the~TikZ~library~'decorations.pathreplacing'.\\
9993     Use: ~\token_to_str:N \usepackage \{tikz\}~
9994     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
9995     That~command~will~be~ignored.
9996 }
9997 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9998 {
9999     Wrong~line.\\
10000     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
10001     \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
10002     number~is~not~valid.~It~will~be~ignored.
10003 }

```

```

10004 \@@_msg_new:nn { Impossible-delimiter }
10005 {
10006     Impossible-delimiter.\\
10007     It's~impossible-to~draw~the~#1~delimiter-of~your~
10008     \token_to_str:N \SubMatrix \ because~all~the~cells~are~empty~
10009     in~that~column.
10010     \bool_if:NT \l_@@_submatrix_slim_bool
10011     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
10012     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
10013 }
10014 \@@_msg_new:nnn { width-without-X-columns }
10015 {
10016     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
10017     the~preamble~(' \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .\\
10018     That~key~will~be~ignored.
10019 }
10020 {
10021     This~message~is~the~message~'width~without~X~columns'~
10022     of~the~module~'nicematrix'.~
10023     The~experimented~users~can~disable~that~message~with~
10024     \token_to_str:N \msg_redirect_name:nnn .\\
10025 }
10026
10027 \@@_msg_new:nn { key-multiplicity-with-dotted }
10028 {
10029     Incompatible~keys. \\
10030     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
10031     in~a~'custom-line'~.~They~are~incompatible. \\
10032     The~key~'multiplicity'~will~be~discarded.
10033 }
10034 \@@_msg_new:nn { empty-environment }
10035 {
10036     Empty~environment.\\
10037     Your~ \@@_full_name_env: \ is~empty.~This~error~is~fatal.
10038 }
10039 \@@_msg_new:nn { No-letter-and-no-command }
10040 {
10041     Erroneous~use.\\
10042     Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
10043     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10044     '~'ccommand'~(to~draw~horizontal~rules).\\
10045     However,~you~can~go~on.
10046 }
10047 \@@_msg_new:nn { Forbidden-letter }
10048 {
10049     Forbidden~letter.\\
10050     You~can't~use~the~letter~'#1'~for~a~customized~line.~
10051     It~will~be~ignored.\\
10052     The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10053 }
10054 \@@_msg_new:nn { Several~letters }
10055 {
10056     Wrong~name.\\
10057     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10058     have~used~' \l_@@_letter_str ').\\
10059     It~will~be~ignored.
10060 }
10061 \@@_msg_new:nn { Delimiter-with-small }
10062 {
10063     Delimiter~forbidden.\\
10064     You~can't~put~a~delimiter~in~the~preamble~of~your~

```

```

10065 \@@_full_name_env: \
10066 because~the~key~'small'~is~in~force.\\
10067 This~error~is~fatal.
10068 }
10069 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10070 {
10071   Unknown~cell.\\
10072   Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10073   the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10074   can't~be~executed~because~a~cell~doesn't~exist.\\
10075   This~command~ \token_to_str:N \line \ will~be~ignored.
10076 }
10077 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10078 {
10079   Duplicate~name.\\
10080   The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10081   in~this~ \@@_full_name_env: .\\
10082   This~key~will~be~ignored.\\
10083   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10084     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10085 }
10086 {
10087   The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
10088   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10089 }
10090 \@@_msg_new:nn { r~or~l~with~preamble }
10091 {
10092   Erroneous~use.\\
10093   You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
10094   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10095   your~ \@@_full_name_env: .\\
10096   This~key~will~be~ignored.
10097 }
10098 \@@_msg_new:nn { Hdotsfor~in~col~0 }
10099 {
10100   Erroneous~use.\\
10101   You~can't~use~ \token_to_str:N \Hdotsfor \ in~an~exterior~column~of~
10102   the~array.~This~error~is~fatal.
10103 }
10104 \@@_msg_new:nn { bad~corner }
10105 {
10106   Bad~corner.\\
10107   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10108   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
10109   This~specification~of~corner~will~be~ignored.
10110 }
10111 \@@_msg_new:nn { bad~border }
10112 {
10113   Bad~border.\\
10114   \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
10115   (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
10116   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10117   also~use~the~key~'tikz'
10118   \IfPackageLoadedF { tikz }
10119     { ~if~you~load~the~LaTeX~package~'tikz' } ).\\
10120   This~specification~of~border~will~be~ignored.
10121 }
10122 \@@_msg_new:nn { TikzEveryCell~without~tikz }
10123 {
10124   TikZ~not~loaded.\\
10125   You~can't~use~ \token_to_str:N \TikzEveryCell \

```

```

10126     because-you-have-not-loaded-tikz.~
10127     This-command-will-be-ignored.
10128 }

10129 \@@_msg_new:nn { tikz-key-without-tikz }
10130 {
10131     TikZ-not-loaded.\\
10132     You-can't-use-the-key-'tikz'~for-the-command~' \token_to_str:N
10133     \Block '~because-you-have-not-loaded-tikz.~
10134     This-key-will-be-ignored.
10135 }

10136 \@@_msg_new:nn { Bad-argument-for-Block }
10137 {
10138     Bad-argument.\\
10139     The-first-mandatory-argument-of~\token_to_str:N \Block\ must~
10140     be-of-the-form~'i-j'~(or-completely-empty)-and-you-have-used:~
10141     '#1'. \\
10142     If-you-go-on,~the~\token_to_str:N \Block\ will-be-mono-cell~(as-if~
10143     the-argument-was-empty).
10144 }

10145 \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
10146 {
10147     Erroneous-use.\\
10148     In-the~ \@@_full_name_env: ,~you-must-use-the-key~
10149     'last-col'~without-value.\\
10150     However,~you-can-go-on-for-this-time~
10151     (the-value~' \l_keys_value_tl '~will-be-ignored).
10152 }

10153 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
10154 {
10155     Erroneous-use. \\
10156     In~\token_to_str:N \NiceMatrixOptions ,~you-must-use-the-key~
10157     'last-col'~without-value. \\
10158     However,~you-can-go-on-for-this-time~
10159     (the-value~' \l_keys_value_tl '~will-be-ignored).
10160 }

10161 \@@_msg_new:nn { Block-too-large-1 }
10162 {
10163     Block-too-large. \\
10164     You-try-to-draw-a-block-in-the-cell-#1-#2-of-your-matrix-but-the-matrix-is~
10165     too-small-for-that-block. \\
10166     This-block-and-maybe-others-will-be-ignored.
10167 }

10168 \@@_msg_new:nn { Block-too-large-2 }
10169 {
10170     Block-too-large. \\
10171     The-preamble-of-your~ \@@_full_name_env: \ announces~ \int_use:N
10172     \g_@@_static_num_of_col_int \
10173     columns~but-you-use-only~ \int_use:N \c@jCol \ and-that's-why-a-block~
10174     specified-in-the-cell-#1-#2-can't-be-drawn.~You-should-add-some-ampersands~
10175     (&)~at-the-end-of-the-first-row-of-your~ \@@_full_name_env: . \\
10176     This-block-and-maybe-others-will-be-ignored.
10177 }

10178 \@@_msg_new:nn { unknown-column-type }
10179 {
10180     Bad-column-type. \\
10181     The-column-type~'#1'~in-your~ \@@_full_name_env: \
10182     is-unknown. \\
10183     This-error-is-fatal.
10184 }

10185 \@@_msg_new:nn { unknown-column-type-multicolumn }

```

```

10186 {
10187   Bad-column-type. \\
10188   The-column-type~'#1'~in~the~command~\token_to_str:N \multicolumn \
10189   ~of~your~ \@@_full_name_env: \
10190   is~unknown. \\
10191   This~error~is~fatal.
10192 }
10193 \@@_msg_new:nn { unknown~column-type-S }
10194 {
10195   Bad-column-type. \\
10196   The-column-type~'S'~in~your~ \@@_full_name_env: \ is~unknown. \\
10197   If~you~want~to~use~the~column-type~'S'~of~siunitx,~you~should~
10198   load~that~package. \\
10199   This~error~is~fatal.
10200 }
10201 \@@_msg_new:nn { unknown~column-type-S-multicolumn }
10202 {
10203   Bad-column-type. \\
10204   The-column-type~'S'~in~the~command~\token_to_str:N \multicolumn \
10205   of~your~ \@@_full_name_env: \ is~unknown. \\
10206   If~you~want~to~use~the~column-type~'S'~of~siunitx,~you~should~
10207   load~that~package. \\
10208   This~error~is~fatal.
10209 }
10210 \@@_msg_new:nn { tabularnote~forbidden }
10211 {
10212   Forbidden-command. \\
10213   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10214   ~here.~This~command~is~available~only~in~
10215   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
10216   the~argument~of~a~command~\token_to_str:N \caption \ included~
10217   in~an~environment~\{table\}. \\
10218   This~command~will~be~ignored.
10219 }
10220 \@@_msg_new:nn { borders~forbidden }
10221 {
10222   Forbidden-key.\\
10223   You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \
10224   because~the~option~'rounded-corners'~
10225   is~in~force~with~a~non-zero~value.\\
10226   This~key~will~be~ignored.
10227 }
10228 \@@_msg_new:nn { bottomrule~without~booktabs }
10229 {
10230   booktabs~not~loaded.\\
10231   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
10232   loaded~'booktabs'.\\
10233   This~key~will~be~ignored.
10234 }
10235 \@@_msg_new:nn { enumitem~not~loaded }
10236 {
10237   enumitem~not~loaded. \\
10238   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10239   ~because~you~haven't~loaded~'enumitem'. \\
10240   All~the~commands~ \token_to_str:N \tabularnote \ will~be~
10241   ignored~in~the~document.
10242 }
10243 \@@_msg_new:nn { tikz~without~tikz }
10244 {
10245   Tikz~not~loaded. \\
10246   You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~

```

```

10247     loaded.~If~you~go~on,~that~key~will~be~ignored.
10248 }

10249 \@@_msg_new:nn { tikz-in-custom-line-without-tikz }
10250 {
10251     Tikz~not~loaded. \\
10252     You~have~used~the~key~'tikz'~in~the~definition~of~a~
10253     customized~line~(with~'custom-line')~but~tikz~is~not~loaded.~
10254     You~can~go~on~but~you~will~have~another~error~if~you~actually~
10255     use~that~custom~line.
10256 }

10257 \@@_msg_new:nn { tikz-in-borders-without-tikz }
10258 {
10259     Tikz~not~loaded. \\
10260     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
10261     command~' \token_to_str:N \Block ')~but~tikz~is~not~loaded.~
10262     That~key~will~be~ignored.
10263 }

10264 \@@_msg_new:nn { color-in-custom-line-with-tikz }
10265 {
10266     Erroneous~use.\\
10267     In~a~'custom-line',~you~have~used~both~'tikz'~and~'color',~
10268     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
10269     The~key~'color'~will~be~discarded.
10270 }

10271 \@@_msg_new:nn { Wrong-last-row }
10272 {
10273     Wrong~number.\\
10274     You~have~used~'last-row= \int_use:N \l_@@_last_row_int '~but~your~
10275     \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
10276     If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
10277     last~row.~You~can~avoid~this~problem~by~using~'last-row'~
10278     without~value~(more~compilations~might~be~necessary).
10279 }

10280 \@@_msg_new:nn { Yet-in-env }
10281 {
10282     Nested~environments.\\
10283     Environments~of~nicematrix~can't~be~nested.\\
10284     This~error~is~fatal.
10285 }

10286 \@@_msg_new:nn { Outside-math-mode }
10287 {
10288     Outside~math~mode.\\
10289     The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
10290     (and~not~in~ \token_to_str:N \vcenter ).\\
10291     This~error~is~fatal.
10292 }

10293 \@@_msg_new:nn { One-letter~allowed }
10294 {
10295     Bad~name.\\
10296     The~value~of~key~' \l_keys_key_str '~must~be~of~length~1~and~
10297     you~have~used~' \l_keys_value_tl '~.\\
10298     It~will~be~ignored.
10299 }

10300 \@@_msg_new:nn { TabularNote-in-CodeAfter }
10301 {
10302     Environment~\{TabularNote\}~forbidden.\\
10303     You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
10304     but~*before*~the~ \token_to_str:N \CodeAfter . \\
10305     This~environment~\{TabularNote\}~will~be~ignored.
10306 }

```

```

10307 \@@_msg_new:nn { varwidth~not~loaded }
10308 {
10309   varwidth~not~loaded.\\
10310   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
10311   loaded.\\
10312   Your~column~will~behave~like~'p'.
10313 }
10314 \@@_msg_new:nn { varwidth~not~loaded~in~X }
10315 {
10316   varwidth~not~loaded.\\
10317   You~can't~use~the~key~'V'~in~your~column~'X'~
10318   because~'varwidth'~is~not~loaded.\\
10319   It~will~be~ignored. \\
10320 }
10321 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
10322 {
10323   Unknown~key.\\
10324   Your~key~' \l_keys_key_str '~is~unknown~for~a~rule.\\
10325   \c_@@_available_keys_str
10326 }
10327 {
10328   The~available~keys~are~(in~alphabetic~order):~
10329   color,~
10330   dotted,~
10331   multiplicity,~
10332   sep-color,~
10333   tikz,~and~total~width.
10334 }
10335
10336 \@@_msg_new:nnn { Unknown~key~for~Block }
10337 {
10338   Unknown~key. \\
10339   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10340   \token_to_str:N \Block . \\
10341   It~will~be~ignored. \\
10342   \c_@@_available_keys_str
10343 }
10344 {
10345   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10346   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line~width,~name,~
10347   opacity,~rounded~corners,~r,~respect~arraystretch,~t,~T,~tikz,~transparent~
10348   and~vlines.
10349 }
10350 \@@_msg_new:nnn { Unknown~key~for~Brace }
10351 {
10352   Unknown~key.\\
10353   The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
10354   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
10355   It~will~be~ignored. \\
10356   \c_@@_available_keys_str
10357 }
10358 {
10359   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
10360   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
10361   right~shorten)~and~yshift.
10362 }
10363 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10364 {
10365   Unknown~key.\\
10366   The~key~' \l_keys_key_str '~is~unknown.\\
10367   It~will~be~ignored. \\
10368   \c_@@_available_keys_str

```

```

10369 }
10370 {
10371     The-available-keys-are~(in~alphabetic~order):~
10372     delimiters/color,~
10373     rules~(with~the~subkeys~'color'~and~'width'),~
10374     sub-matrix~(several~subkeys)~
10375     and~xdots~(several~subkeys).~
10376     The~latter~is~for~the~command~ \token_to_str:N \line .
10377 }
10378 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10379 {
10380     Unknown~key.\\
10381     The~key~' \l_keys_key_str '~is~unknown.\\
10382     It~will~be~ignored. \\
10383     \c_@@_available_keys_str
10384 }
10385 {
10386     The-available-keys-are~(in~alphabetic~order):~
10387     create-cell-nodes,~
10388     delimiters/color~and~
10389     sub-matrix~(several~subkeys).
10390 }
10391 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10392 {
10393     Unknown~key.\\
10394     The~key~' \l_keys_key_str '~is~unknown.\\
10395     That~key~will~be~ignored. \\
10396     \c_@@_available_keys_str
10397 }
10398 {
10399     The-available-keys-are~(in~alphabetic~order):~
10400     'delimiters/color',~
10401     'extra-height',~
10402     'hlines',~
10403     'hvlines',~
10404     'left-xshift',~
10405     'name',~
10406     'right-xshift',~
10407     'rules'~(with~the~subkeys~'color'~and~'width'),~
10408     'slim',~
10409     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10410     and~'right-xshift').\\
10411 }
10412 \@@_msg_new:nnn { Unknown~key~for~notes }
10413 {
10414     Unknown~key.\\
10415     The~key~' \l_keys_key_str '~is~unknown.\\
10416     That~key~will~be~ignored. \\
10417     \c_@@_available_keys_str
10418 }
10419 {
10420     The-available-keys-are~(in~alphabetic~order):~
10421     bottomrule,~
10422     code-after,~
10423     code-before,~
10424     detect-duplicates,~
10425     enumitem-keys,~
10426     enumitem-keys-para,~
10427     para,~
10428     label-in-list,~
10429     label-in-tabular~and~
10430     style.
10431 }

```



```

10432 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10433 {
10434   Unknown~key.\\
10435   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10436   \token_to_str:N \RowStyle . \\
10437   That~key~will~be~ignored. \\
10438   \c_@@_available_keys_str
10439 }
10440 {
10441   The~available~keys~are~(in~alphabetic~order):~
10442   bold,~
10443   cell-space-top-limit,~
10444   cell-space-bottom-limit,~
10445   cell-space-limits,~
10446   color,~
10447   fill~(alias:~rowcolor),~
10448   nb-rows,~
10449   opacity~and~
10450   rounded-corners.
10451 }
10452 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10453 {
10454   Unknown~key.\\
10455   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10456   \token_to_str:N \NiceMatrixOptions . \\
10457   That~key~will~be~ignored. \\
10458   \c_@@_available_keys_str
10459 }
10460 {
10461   The~available~keys~are~(in~alphabetic~order):~
10462   &~in~blocks,~
10463   allow-duplicate-names,~
10464   ampersand-in-blocks,~
10465   caption-above,~
10466   cell-space-bottom-limit,~
10467   cell-space-limits,~
10468   cell-space-top-limit,~
10469   code-for-first-col,~
10470   code-for-first-row,~
10471   code-for-last-col,~
10472   code-for-last-row,~
10473   corners,~
10474   custom-key,~
10475   create-extra-nodes,~
10476   create-medium-nodes,~
10477   create-large-nodes,~
10478   custom-line,~
10479   delimiters~(several~subkeys),~
10480   end-of-row,~
10481   first-col,~
10482   first-row,~
10483   hlines,~
10484   hvlines,~
10485   hvlines-except-borders,~
10486   last-col,~
10487   last-row,~
10488   left-margin,~
10489   light-syntax,~
10490   light-syntax-expanded,~
10491   matrix/columns-type,~
10492   no-cell-nodes,~
10493   notes~(several~subkeys),~
10494   nullify-dots,~

```

```

10495 pgf-node-code,~
10496 renew-dots,~
10497 renew-matrix,~
10498 respect-arraystretch,~
10499 rounded-corners,~
10500 right-margin,~
10501 rules~(with~the~subkeys~'color'~and~'width'),~
10502 small,~
10503 sub-matrix~(several~subkeys),~
10504 vl原因,~
10505 xdots~(several~subkeys).
10506 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

10507 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10508 {
10509   Unknown~key.\\
10510   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
10511   \{NiceArray\}. \\
10512   That~key~will~be~ignored. \\
10513   \c_@@_available_keys_str
10514 }
10515 {
10516   The~available~keys~are~(in~alphabetic~order):~
10517   &-in-blocks,~
10518   ampersand-in-blocks,~
10519   b,~
10520   baseline,~
10521   c,~
10522   cell-space-bottom-limit,~
10523   cell-space-limits,~
10524   cell-space-top-limit,~
10525   code-after,~
10526   code-for-first-col,~
10527   code-for-first-row,~
10528   code-for-last-col,~
10529   code-for-last-row,~
10530   columns-width,~
10531   corners,~
10532   create-extra-nodes,~
10533   create-medium-nodes,~
10534   create-large-nodes,~
10535   extra-left-margin,~
10536   extra-right-margin,~
10537   first-col,~
10538   first-row,~
10539   hlines,~
10540   hvlines,~
10541   hvlines-except-borders,~
10542   last-col,~
10543   last-row,~
10544   left-margin,~
10545   light-syntax,~
10546   light-syntax-expanded,~
10547   name,~
10548   no-cell-nodes,~
10549   nullify-dots,~
10550   pgf-node-code,~
10551   renew-dots,~
10552   respect-arraystretch,~
10553   right-margin,~
10554   rounded-corners,~
10555   rules~(with~the~subkeys~'color'~and~'width'),~

```

```

10556     small,~
10557     t,~
10558     vlines,~
10559     xdots/color,~
10560     xdots/shorten-start,~
10561     xdots/shorten-end,~
10562     xdots/shorten-and~
10563     xdots/line-style.
10564 }

```

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`).

```

10565 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10566 {
10567     Unknown~key.\\
10568     The~key~' \l_keys_key_str '~is~unknown~for~the~
10569     \@@_full_name_env: . \\
10570     That~key~will~be~ignored. \\
10571     \c_@@_available_keys_str
10572 }
10573 {
10574     The~available~keys~are~(in~alphabetic~order):~
10575     &~in~blocks,~
10576     ampersand~in~blocks,~
10577     b,~
10578     baseline,~
10579     c,~
10580     cell-space-bottom-limit,~
10581     cell-space-limits,~
10582     cell-space-top-limit,~
10583     code~after,~
10584     code~for~first~col,~
10585     code~for~first~row,~
10586     code~for~last~col,~
10587     code~for~last~row,~
10588     columns~type,~
10589     columns~width,~
10590     corners,~
10591     create~extra~nodes,~
10592     create~medium~nodes,~
10593     create~large~nodes,~
10594     extra~left~margin,~
10595     extra~right~margin,~
10596     first~col,~
10597     first~row,~
10598     hlines,~
10599     hvlines,~
10600     hvlines~except~borders,~
10601     l,~
10602     last~col,~
10603     last~row,~
10604     left~margin,~
10605     light~syntax,~
10606     light~syntax~expanded,~
10607     name,~
10608     no~cell~nodes,~
10609     nullify~dots,~
10610     pgf~node~code,~
10611     r,~
10612     renew~dots,~
10613     respect~arraystretch,~
10614     right~margin,~
10615     rounded~corners,~
10616     rules~(with~the~subkeys~'color'~and~'width'),~

```

```

10617     small,~
10618     t,~
10619     vlines,~
10620     xdots/color,~
10621     xdots/shorten-start,~
10622     xdots/shorten-end,~
10623     xdots/shorten-and~
10624     xdots/line-style.
10625 }
10626 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10627 {
10628     Unknown~key.\\
10629     The~key~' \l_keys_key_str 'is~unknown~for~the~environment~
10630     \{NiceTabular\}. \\
10631     That~key~will~be~ignored. \\
10632     \c_@@_available_keys_str
10633 }
10634 {
10635     The~available~keys~are~(in~alphabetic~order):~
10636     &-in-blocks,~
10637     ampersand-in-blocks,~
10638     b,~
10639     baseline,~
10640     c,~
10641     caption,~
10642     cell-space-bottom-limit,~
10643     cell-space-limits,~
10644     cell-space-top-limit,~
10645     code-after,~
10646     code-for-first-col,~
10647     code-for-first-row,~
10648     code-for-last-col,~
10649     code-for-last-row,~
10650     columns-width,~
10651     corners,~
10652     custom-line,~
10653     create-extra-nodes,~
10654     create-medium-nodes,~
10655     create-large-nodes,~
10656     extra-left-margin,~
10657     extra-right-margin,~
10658     first-col,~
10659     first-row,~
10660     hlines,~
10661     hvlines,~
10662     hvlines-except-borders,~
10663     label,~
10664     last-col,~
10665     last-row,~
10666     left-margin,~
10667     light-syntax,~
10668     light-syntax-expanded,~
10669     name,~
10670     no-cell-nodes,~
10671     notes~(several~subkeys),~
10672     nullify-dots,~
10673     pgf-node-code,~
10674     renew-dots,~
10675     respect-arraystretch,~
10676     right-margin,~
10677     rounded-corners,~
10678     rules~(with~the~subkeys~'color'~and~'width'),~
10679     short-caption,~

```

```

10680     t,~
10681     tabularnote,~
10682     vlines,~
10683     xdots/color,~
10684     xdots/shorten-start,~
10685     xdots/shorten-end,~
10686     xdots/shorten-and~
10687     xdots/line-style.
10688 }
10689 \@@_msg_new:nnn { Duplicate-name }
10690 {
10691   Duplicate-name.\\
10692   The-name~' \l_keys_value_tl '~is-already-used-and-you-shouldn't-use~
10693   the-same-environment-name-twice.~You-can-go-on,~but,~
10694   maybe,~you-will-have-incorrect-results-especially~
10695   if-you-use~'columns-width=auto'.~If-you-don't-want-to-see-this~
10696   message-again,~use-the-key~'allow-duplicate-names'~in~
10697   ' \token_to_str:N \NiceMatrixOptions '.\\
10698   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10699     { For-a-list-of-the-names-already-used,~type-H~<return>. }
10700 }
10701 {
10702   The-names-already-defined-in-this-document-are:~
10703   \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
10704 }
10705 \@@_msg_new:nn { Option-auto-for-columns-width }
10706 {
10707   Erroneous-use.\\
10708   You-can't-give-the-value~'auto'~to-the-key~'columns-width'~here.~
10709   That-key-will-be-ignored.
10710 }
10711 \@@_msg_new:nn { NiceTabularX-without~X }
10712 {
10713   NiceTabularX-without~X.\\
10714   You-should-not-use~\{NiceTabularX\}~without~X~columns.\\
10715   However,~you-can-go-on.
10716 }
10717 \@@_msg_new:nn { Preamble-forgotten }
10718 {
10719   Preamble-forgotten.\\
10720   You-have-probably-forgotten-the-preamble-of-your~
10721   \@@_full_name_env: . \\
10722   This-error-is-fatal.
10723 }
10724 \@@_msg_new:nn { Invalid-col-number }
10725 {
10726   Invalid-column-number.\\
10727   A-color-instruction-in-the~ \token_to_str:N \CodeBefore \
10728   specifies-a-column-which-is-outside-the-array.~It-will-be-ignored.
10729 }
10730 \@@_msg_new:nn { Invalid-row-number }
10731 {
10732   Invalid-row-number.\\
10733   A-color-instruction-in-the~ \token_to_str:N \CodeBefore \
10734   specifies-a-row-which-is-outside-the-array.~It-will-be-ignored.
10735 }
10736 \@@_define_com:NNN p ( )
10737 \@@_define_com:NNN b [ ]
10738 \@@_define_com:NNN v | |
10739 \@@_define_com:NNN V \l \l
10740 \@@_define_com:NNN B \{ \}

```

Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	3
4	Parameters	9
5	The command <code>\tabularnote</code>	20
6	Command for creation of rectangle nodes	24
7	The options	25
8	Important code used by <code>{NiceArrayWithDelims}</code>	36
9	The <code>\CodeBefore</code>	50
10	The environment <code>{NiceArrayWithDelims}</code>	54
11	Construction of the preamble of the array	60
12	The redefinition of <code>\multicolumn</code>	75
13	The environment <code>{NiceMatrix}</code> and its variants	93
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	94
15	After the construction of the array	95
16	We draw the dotted lines	102
17	The actual instructions for drawing the dotted lines with Tikz	117
18	User commands available in the new environments	122
19	The command <code>\line</code> accessible in code-after	128
20	The command <code>\RowStyle</code>	130
21	Colors of cells, rows and columns	133
22	The vertical and horizontal rules	145
23	The empty corners	161
24	The environment <code>{NiceMatrixBlock}</code>	164
25	The extra nodes	165
26	The blocks	169
27	How to draw the dotted lines transparently	194
28	Automatic arrays	194
29	The redefinition of the command <code>\dotfill</code>	196
30	The command <code>\diagbox</code>	196

31	The keyword <code>\CodeAfter</code>	197
32	The delimiters in the preamble	198
33	The command <code>\SubMatrix</code>	199
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	208
35	The commands <code>HBrace</code> et <code>VBrace</code>	211
36	The command <code>TikzEveryCell</code>	214
37	The command <code>\ShowCellNames</code>	216
38	We process the options at package loading	217
39	About the package underscore	219
40	Error messages of the package	219