

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

January 5, 2026

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

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

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

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

*This document corresponds to the version 7.5a of `nicematrix`, at the date of 2026/01/05.

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

```

20 \RequirePackage { amsmath }

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

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

```

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

```

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

```

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

```

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

```

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

```

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

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

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

```

2 Collecting options

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

Example :

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

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

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

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

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

```

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

```

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

```

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

```

3 Technical definitions

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

```

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

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

```

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

```

83 \tl_new:N \l_@@_argspec_tl

```

```

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

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

We define a command `\iddots` similar to `\ddots` (``) 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.

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

171     }
172 }

```

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

```

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

```

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

```

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

```

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

```

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

```

Idem for `\CT@drs@`.

```

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

```

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

```

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

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

¹See question 99041 on TeX StackExchange.

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```


4 Parameters

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Idem for the mono-row blocks.

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

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

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

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

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

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

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

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

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

```

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

```

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

```

327 \dim_new:N \l_@@_tabular_width_dim

```

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

```

328 \dim_new:N \l_@@_rule_width_dim

```

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

```

329 \tl_new:N \l_@@_rule_color_tl

```

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

```

330 \bool_new:N \g_@@_rotate_bool

```

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

```

331 \bool_new:N \g_@@_rotate_c_bool

```

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

```

332 \bool_new:N \l_@@_X_bool

```

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

```

333 \bool_new:N \l_@@_V_of_X_bool

```

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

```

334 \bool_new:N \g_@@_V_of_X_bool

```

```

335 \bool_new:N \g_@@_caption_finished_bool

```

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

```

336 \bool_new:N \l_@@_no_cell_nodes_bool

```

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

```

337 \tl_new:N \g_@@_aux_tl

```

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

```

338 \bool_new:N \g_@@_aux_found_bool

```

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

```

339 \seq_new:N \g_@@_size_seq

```

```

340 \tl_new:N \g_@@_left_delim_tl

```

```

341 \tl_new:N \g_@@_right_delim_tl

```

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

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

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

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

For `\multicolumn`.

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

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

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

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

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

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

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

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

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

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

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

```
352 {
```

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

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

```
355   \fi:
```

```
356 }
```

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

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

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

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

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

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

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

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

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

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

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

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)`.

```

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

```

```

370 \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.

```

371 \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.

```

372 \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.

```

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

```

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

```

377 \bool_new:N \l_@@_in_code_after_bool

```

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

```

378 \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.

```

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

```

381 \seq_new:N \l_@@_custom_line_commands_seq

```

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

```

382 \tl_new:N \l_@@_rules_color_tl

```

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

```
383 \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.

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

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

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

```
387 \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).

```
388 \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).

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

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

```
391 \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).

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

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

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

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

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

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

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

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

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

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

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

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

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

```
403 \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”.

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

```
405 \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.

```
406 \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!).

```
407 \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.

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

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

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

```
409 \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.

```
410 \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`).

```
411 \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`).

```
412 \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.

```
413 \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).

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

```
415 \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.

```
416 \int_new:N \g_@@_ddots_int
417 \int_new:N \g_@@_iddots_int
```

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

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

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

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

The following counters will be used when drawing the rules.

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

```
435 \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).

```
436 \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`.

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

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

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

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

```
443 \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`.

```
444 \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.

```
445 \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.

```
446 \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`).

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

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

```
451 \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).

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

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

```
453 \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).

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

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

457 \dim_new:N \l_@@_submatrix_extra_height_dim
458 \dim_new:N \l_@@_submatrix_left_xshift_dim
459 \dim_new:N \l_@@_submatrix_right_xshift_dim
460 \clist_new:N \l_@@_hlines_clist
461 \clist_new:N \l_@@_vlines_clist
462 \clist_new:N \l_@@_submatrix_hlines_clist
463 \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}`).

```
464 \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.

```
465 \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`).

```
466 \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.

```
467 \int_new:N \l_@@_first_row_int
468 \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.

```
469 \int_new:N \l_@@_first_col_int
470 \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).

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

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

Idem for `\l_@@_last_col_without_value_bool`

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

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

- **Last column**

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

```
475 \int_new:N \l_@@_last_col_int
476 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

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

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

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

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

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

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

Some utilities

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

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

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

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

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

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

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

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

```

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

```

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

```

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

```

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.

```

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

```

5 The command `\tabularnote`

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

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

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

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

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

```
518 \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`.

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

```
523 \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.

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

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

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

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

```
532 \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`.

```
533 \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`.

```
534 \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`).

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

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.

```

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

```

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`.

```

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

```

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

```

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

```

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).

```

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

```

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

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

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

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

```

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

```

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`.

```

628     \hbox_set:Nn \l_tmpa_box
629     {

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```



```

666         { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
667     }
Now, we will compose the label of the footnote (in the caption). Even if we are not in the first
composition, we have to compose that label!
668     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
669     \seq_put_right:Ne \l_@@_notes_labels_seq
670     {
671         \tl_if_novalue:nTF { #1 }
672         { \@@_notes_format:n { \int_use:N \c@tabularnote } }
673         { #1 }
674     }
675     \peek_meaning:NF \tabularnote
676     {
677         \@@_notes_label_in_tabular:n { \seq_use:Nn \l_@@_notes_labels_seq { , } }
678         \seq_clear:N \l_@@_notes_labels_seq
679     }
680 }
681 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
682 { \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.

```

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

```

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.

```

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

```

```

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

```

7 The options

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

```

730 \tl_new:N \l_@@_caption_tl
731 \tl_new:N \l_@@_short_caption_tl
732 \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).

```

733 \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`.

```

734 \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`).

```

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

```

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

```

737 \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.

```

738 \dim_new:N \l_@@_xdots_inter_dim
739 \hook_gput_code:nnn { begindocument } { . }
740 { \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).

```

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

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.

```

748 \dim_new:N \l_@@_xdots_radius_dim
749 \hook_gput_code:nnn { begindocument } { . }
750 { \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.

```

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

```

754 \bool_new:N \l_@@_light_syntax_bool
755 \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).

```

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

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

```

758 \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`).

```

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

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

```

760 \bool_new:N \l_@@_parallelize_diags_bool
761 \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.

```

762 \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.).

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

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

```
764 \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).

```
765 \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!

```
766 \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.

```
767 \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”.

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

```
769 \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).

```
770 \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`).

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

```
772 \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`.

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

```
774 \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.

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

```
776 \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 “:”.

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

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

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

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

```

779 \bool_new:N \l_@@_delimiters_max_width_bool

780 \keys_define:nn { nicematrix / xdots }
781 {
782   nullify .bool_set:N = \l_@@_nullify_dots_bool ,
783   nullify .default:n = true ,
784   brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
785   brace-shift .value_required:n = true ,
786   Vbrace .bool_set:N = \l_@@_Vbrace_bool ,
787   shorten-start .code:n =
788     \hook_gput_code:nnn { begindocument } { . }
789     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
790   shorten-end .code:n =
791     \hook_gput_code:nnn { begindocument } { . }
792     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
793   shorten-start .value_required:n = true ,
794   shorten-end .value_required:n = true ,
795   shorten .code:n =
796     \hook_gput_code:nnn { begindocument } { . }
797     {
798       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
799       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
800     } ,
801   shorten .value_required:n = true ,
802   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
803   horizontal-labels .default:n = true ,
804   horizontal-label .bool_set:N = \l_@@_xdots_h_labels_bool ,
805   horizontal-label .default:n = true ,
806   line-style .code:n =
807     {
808       \bool_lazy_or:nnTF
809         { \cs_if_exist_p:N \tikzpicture }
810         { \str_if_eq_p:nn { #1 } { standard } }
811         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
812         { \@@_error:n { bad-option-for-line-style } }
813     } ,
814   line-style .value_required:n = true ,
815   color .tl_set:N = \l_@@_xdots_color_tl ,
816   color .value_required:n = true ,
817   radius .code:n =
818     \hook_gput_code:nnn { begindocument } { . }
819     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
820   radius .value_required:n = true ,
821   inter .code:n =
822     \hook_gput_code:nnn { begindocument } { . }
823     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
824   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}`.

```

825   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
826   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
827   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).

```

828     draw-first .code:n = \prg_do_nothing: ,
829     unknown .code:n =
830       \@@_unknown_key:nn { nicematrix / xdots } { Unknown-key-for-xdots }
831   }

832 \keys_define:nn { nicematrix / rules }
833 {
834   color .tl_set:N = \l_@@_rules_color_tl ,
835   color .value_required:n = true ,
836   width .dim_set:N = \arrayrulewidth ,
837   width .value_required:n = true ,
838   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
839 }

```

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

```

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

```

```

882 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
883 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
884 last-row .int_set:N = \l_@@_last_row_int ,
885 last-row .default:n = -1 ,
886 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
887 code-for-first-col .value_required:n = true ,
888 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
889 code-for-last-col .value_required:n = true ,
890 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
891 code-for-first-row .value_required:n = true ,
892 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
893 code-for-last-row .value_required:n = true ,
894 hlines .clist_set:N = \l_@@_hlines_clist ,
895 vlines .clist_set:N = \l_@@_vlines_clist ,
896 hlines .default:n = all ,
897 vlines .default:n = all ,
898 vlines-in-sub-matrix .code:n =
899 {
900   \tl_if_single_token:nTF { #1 }
901   {
902     \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
903     { \@@_error:nn { Forbidden-letter } { #1 } }

```

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

```

904     { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
905   }
906   { \@@_error:n { One-letter-allowed } }
907 },
908 vlines-in-sub-matrix .value_required:n = true ,
909 hvlines .code:n =
910 {
911   \bool_set_true:N \l_@@_hvlines_bool
912   \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
913   \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
914 },
915 hvlines .value_forbidden:n = true ,
916 hvlines-except-borders .code:n =
917 {
918   \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
919   \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
920   \bool_set_true:N \l_@@_hvlines_bool
921   \bool_set_true:N \l_@@_except_borders_bool
922 },
923 hvlines-except-borders .value_forbidden:n = true ,
924 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.

```

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

```

```

940     extra-margin .meta:n =
941     { extra-left-margin = #1 , extra-right-margin = #1 } ,
942     extra-margin .value_required:n = true ,
943     respect-arraystretch .code:n =
944     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
945     respect-arraystretch .value_forbidden:n = true ,
946     pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
947     pgf-node-code .value_required:n = true
948 }

```

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

```

949 \keys_define:nn { nicematrix / environments }
950 {
951     corners .clist_set:N = \l_@@_corners_clist ,
952     corners .default:n = { NW , SW , NE , SE } ,
953     code-before .code:n =
954     {
955         \tl_if_empty:nF { #1 }
956         {
957             \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
958             \bool_set_true:N \l_@@_code_before_bool
959         }
960     } ,
961     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}`.

```

962     c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
963     t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
964     b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
965     baseline .tl_set:N = \l_@@_baseline_tl ,
966     baseline .value_required:n = true ,
967     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)`.

```

968     \str_if_eq:eeTF { #1 } { auto }
969     { \bool_set_true:N \l_@@_auto_columns_width_bool }
970     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
971     columns-width .value_required:n = true ,
972     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.

```

973     \legacy_if:nF { measuring@ }
974     {
975         \str_set:Ne \l_@@_name_str { #1 }
976         \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
977         { \@@_err_duplicate_names:n { #1 } }
978         { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
979     } ,
980     name .value_required:n = true ,
981     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
982     code-after .value_required:n = true ,
983 }

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

986 \keys_define:nn { nicematrix / notes }
987 {
988     para .bool_set:N = \l_@@_notes_para_bool ,
989     para .default:n = true ,

```



```

990   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
991   code-before .value_required:n = true ,
992   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
993   code-after .value_required:n = true ,
994   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
995   bottomrule .default:n = true ,
996   style .cs_set:Np = \@@_notes_style:n #1 ,
997   style .value_required:n = true ,
998   label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
999   label-in-tabular .value_required:n = true ,
1000  label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1001  label-in-list .value_required:n = true ,
1002  enumitem-keys .code:n =
1003  {
1004    \hook_gput_code:nnn { begindocument } { . }
1005    {
1006      \IfPackageLoadedT { enumitem }
1007      { \setlist* [ tabularnotes ] { #1 } }
1008    }
1009  } ,
1010  enumitem-keys .value_required:n = true ,
1011  enumitem-keys-para .code:n =
1012  {
1013    \hook_gput_code:nnn { begindocument } { . }
1014    {
1015      \IfPackageLoadedT { enumitem }
1016      { \setlist* [ tabularnotes* ] { #1 } }
1017    }
1018  } ,
1019  enumitem-keys-para .value_required:n = true ,
1020  detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1021  detect-duplicates .default:n = true ,
1022  unknown .code:n =
1023  \@@_unknown_key:nn { nicematrix / notes } { Unknown~key~for~notes }
1024  }
1025  \keys_define:nn { nicematrix / delimiters }
1026  {
1027    max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1028    max-width .default:n = true ,
1029    color .tl_set:N = \l_@@_delimiters_color_tl ,
1030    color .value_required:n = true ,
1031  }

```

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

```

1032  \keys_define:nn { nicematrix }
1033  {
1034    NiceMatrixOptions .inherit:n =
1035    { nicematrix / Global } ,
1036    NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1037    NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1038    NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1039    NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1040    SubMatrix / rules .inherit:n = nicematrix / rules ,
1041    CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1042    CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1043    CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1044    NiceMatrix .inherit:n =
1045    {
1046      nicematrix / Global ,
1047      nicematrix / environments ,
1048    } ,

```

```

1049   NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1050   NiceMatrix / rules .inherit:n = nicematrix / rules ,
1051   NiceTabular .inherit:n =
1052   {
1053       nicematrix / Global ,
1054       nicematrix / environments
1055   } ,
1056   NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1057   NiceTabular / rules .inherit:n = nicematrix / rules ,
1058   NiceTabular / notes .inherit:n = nicematrix / notes ,
1059   NiceArray .inherit:n =
1060   {
1061       nicematrix / Global ,
1062       nicematrix / environments ,
1063   } ,
1064   NiceArray / xdots .inherit:n = nicematrix / xdots ,
1065   NiceArray / rules .inherit:n = nicematrix / rules ,
1066   pNiceArray .inherit:n =
1067   {
1068       nicematrix / Global ,
1069       nicematrix / environments ,
1070   } ,
1071   pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1072   pNiceArray / rules .inherit:n = nicematrix / rules ,
1073 }

```

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

```

1074 \keys_define:nn { nicematrix / NiceMatrixOptions }
1075 {
1076   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1077   delimiters / color .value_required:n = true ,
1078   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1079   delimiters / max-width .default:n = true ,
1080   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1081   delimiters .value_required:n = true ,
1082   width .dim_set:N = \l_@@_width_dim ,
1083   width .value_required:n = true ,
1084   last-col .code:n =
1085     \tl_if_empty:nF { #1 }
1086     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1087     \int_zero:N \l_@@_last_col_int ,
1088   small .bool_set:N = \l_@@_small_bool ,
1089   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.

```

1090   renew-matrix .code:n = \@@_renew_matrix: ,
1091   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}`.

```

1092   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.

```

1093   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)`.

```

1094   \str_if_eq:eeTF { #1 } { auto }
1095   { \@@_error:n { Option~auto~for~columns-width } }
1096   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1097   allow-duplicate-names .code:n =
1098     \cs_set:Nn \@@_err_duplicate_names:n { } ,
1099   allow-duplicate-names .value_forbidden:n = true ,
1100   notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1101   notes .value_required:n = true ,
1102   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1103   sub-matrix .value_required:n = true ,
1104   matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1105   matrix / columns-type .value_required:n = true ,
1106   caption-above .bool_set:N = \l_@@_caption_above_bool ,
1107   caption-above .default:n = true ,
1108   unknown .code:n =
1109     \@@_unknown_key:nn
1110       { nicematrix / Global , nicematrix / NiceMatrixOptions }
1111       { Unknown-key-for-NiceMatrixOptions }
1112 }

```

`\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.

```

1113 \NewDocumentCommand \NiceMatrixOptions { m }
1114 { \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.

```

1115 \keys_define:nn { nicematrix / NiceMatrix }
1116 {
1117   last-col .code:n = \tl_if_empty:nTF { #1 }
1118     {
1119       \bool_set_true:N \l_@@_last_col_without_value_bool
1120       \int_set:Nn \l_@@_last_col_int { -1 }
1121     }
1122     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1123   columns-type .tl_set:N = \l_@@_columns_type_tl ,
1124   columns-type .value_required:n = true ,
1125   l .meta:n = { columns-type = l } ,
1126   r .meta:n = { columns-type = r } ,
1127   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1128   delimiters / color .value_required:n = true ,
1129   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1130   delimiters / max-width .default:n = true ,
1131   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1132   delimiters .value_required:n = true ,
1133   small .bool_set:N = \l_@@_small_bool ,
1134   small .value_forbidden:n = true ,
1135   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1136 }

```

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

```

1137 \keys_define:nn { nicematrix / NiceArray }
1138 {

```

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.

```

1139   small .bool_set:N = \l_@@_small_bool ,
1140   small .value_forbidden:n = true ,
1141   last-col .code:n = \tl_if_empty:nF { #1 }
1142     { \@@_error:n { last-col-non-empty-for-NiceArray } }

```

```

1143         \int_zero:N \l_@@_last_col_int ,
1144     r .code:n = \@@_error:n { r-or~l-with~preamble } ,
1145     l .code:n = \@@_error:n { r-or~l-with~preamble } ,
1146     unknown .code:n =
1147         \@@_unknown_key:nn
1148         { nicematrix / NiceArray , nicematrix / Global , nicematrix / environments}
1149         { Unknown-key~for~NiceArray }
1150 }
1151 \keys_define:nn { nicematrix / pNiceArray }
1152 {
1153     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1154     last-col .code:n = \tl_if_empty:nF { #1 }
1155         { \@@_error:n { last-col-non-empty-for~NiceArray } }
1156         \int_zero:N \l_@@_last_col_int ,
1157     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1158     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1159     delimiters / color .value_required:n = true ,
1160     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1161     delimiters / max-width .default:n = true ,
1162     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1163     delimiters .value_required:n = true ,
1164     small .bool_set:N = \l_@@_small_bool ,
1165     small .value_forbidden:n = true ,
1166     r .code:n = \@@_error:n { r-or~l-with~preamble } ,
1167     l .code:n = \@@_error:n { r-or~l-with~preamble } ,
1168     unknown .code:n =
1169         \@@_unknown_key:nn
1170         { nicematrix / pNiceArray , nicematrix / Global , nicematrix / environments }
1171         { Unknown-key~for~NiceMatrix }
1172 }

```

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

```

1173 \keys_define:nn { nicematrix / NiceTabular }
1174 {

```

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.

```

1175     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1176         \bool_set_true:N \l_@@_width_used_bool ,
1177     width .value_required:n = true ,
1178     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1179     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1180     tabularnote .value_required:n = true ,
1181     caption .tl_set:N = \l_@@_caption_tl ,
1182     caption .value_required:n = true ,
1183     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1184     short-caption .value_required:n = true ,
1185     label .tl_set:N = \l_@@_label_tl ,
1186     label .value_required:n = true ,
1187     last-col .code:n = \tl_if_empty:nF { #1 }
1188         { \@@_error:n { last-col-non-empty-for~NiceArray } }
1189         \int_zero:N \l_@@_last_col_int ,
1190     r .code:n = \@@_error:n { r-or~l-with~preamble } ,
1191     l .code:n = \@@_error:n { r-or~l-with~preamble } ,
1192     unknown .code:n =
1193         \@@_unknown_key:nn
1194         { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
1195         { Unknown-key~for~NiceTabular }
1196 }

```

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
```

```
1197 \keys_define:nn { nicematrix / CodeAfter }
1198 {
1199   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1200   delimiters / color .value_required:n = true ,
1201   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1202   rules .value_required:n = true ,
1203   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1204   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1205   sub-matrix .value_required:n = true ,
1206   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1207 }
```

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}`).

```
1208 \cs_new_protected:Npn \@@_cell_begin:
1209 {
```

`\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.

```
1210   \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).

```
1211   \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.

```
1212   \cs_set_eq:NN \Hline \@@_Hline_in_cell:
```

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

```
1213   \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.

```
1214   \int_compare:nNnT { \c@jCol } = { \c_one_int }
1215   {
1216     \int_compare:nNnT { \l_@@_first_col_int } = { \c_one_int }
1217     { \@@_begin_of_row: }
1218   }
```

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:`.

```
1219   \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1220   \@@_tuning_not_tabular_begin:
1221   \@@_tuning_first_row:
1222   \@@_tuning_last_row:
1223   \g_@@_row_style_tl
1224 }
```

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.

```
1225 \cs_new_protected:Npn \@@_tuning_first_row:
1226 {
1227   \if_int_compare:w \c@iRow = \c_zero_int
1228     \if_int_compare:w \c@jCol > \c_zero_int
1229       \l_@@_code_for_first_row_tl
1230       \xglobal \colorlet { nicematrix-first-row } { . }
1231     \fi:
1232   \fi:
1233 }
```

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

```
\cs_new_protected:Npn \@@_tuning_last_row:
{
  \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
  {
    \l_@@_code_for_last_row_tl
    \xglobal \colorlet { nicematrix-last-row } { . }
  }
}
```

We will use a version a little more efficient.

```
1234 \cs_new_protected:Npn \@@_tuning_last_row:
1235 {
1236   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1237     \l_@@_code_for_last_row_tl
1238     \xglobal \colorlet { nicematrix-last-row } { . }
1239   \fi:
1240 }
```

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

```
1241 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```
1242 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1243 {
1244   \m@th
1245   $ % $
```

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

```
1246 \@@_tuning_key_small:
1247 }
1248 \cs_set:Nn \@@_tuning_not_tabular_end: { $ } % $
```

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

```

1249 \cs_new_protected:Npn \@@_begin_of_row:
1250 {
1251   \int_gincr:N \c@iRow
1252   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1253   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1254   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1255   \pgfpicture
1256   \pgfrememberpicturepositiononpagetrue
1257   \pgfcoordinate
1258   { \@@_env: - row - \int_use:N \c@iRow - base }
1259   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1260   \str_if_empty:NF \l_@@_name_str
1261   {
1262     \pgfnodealias
1263     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1264     { \@@_env: - row - \int_use:N \c@iRow - base }
1265   }
1266   \endpgfpicture
1267 }

```

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.

```

1268 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1269 {
1270   \int_if_zero:nTF { \c@iRow }
1271   {
1272     \dim_compare:nNnT
1273     { \box_dp:N \l_@@_cell_box } > { \g_@@_dp_row_zero_dim }
1274     { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1275     \dim_compare:nNnT
1276     { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_zero_dim }
1277     { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1278   }
1279   {
1280     \int_compare:nNnT { \c@iRow } = { \c_one_int }
1281     {
1282       \dim_compare:nNnT
1283       { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_one_dim }
1284       { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1285     }
1286   }
1287 }
1288 \cs_new_protected:Npn \@@_rotate_cell_box:
1289 {
1290   \box_rotate:Nn \l_@@_cell_box { 90 }
1291   \bool_if:NTF \g_@@_rotate_c_bool
1292   {
1293     \hbox_set:Nn \l_@@_cell_box
1294     {
1295       \m@th
1296       $ % $
1297       \vcenter { \box_use:N \l_@@_cell_box }
1298       $ % $
1299     }
1300   }
1301 }

```

```

1302     \int_compare:nNt { \c@iRow } = { \l_@@_last_row_int }
1303     {
1304         \vbox_set_top:Nn \l_@@_cell_box
1305         {
1306             \vbox_to_zero:n { }
1307             \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1308             \box_use:N \l_@@_cell_box
1309         }
1310     }
1311 }
1312 \bool_gset_false:N \g_@@_rotate_bool
1313 \bool_gset_false:N \g_@@_rotate_c_bool
1314 }
1315 \cs_new_protected:Npn \@@_adjust_size_box:
1316 {
1317     \dim_compare:nNt { \g_@@_blocks_wd_dim } > { \c_zero_dim }
1318     {
1319         \box_set_wd:Nn \l_@@_cell_box
1320         { \dim_max:nn { \box_wd:N \l_@@_cell_box } { \g_@@_blocks_wd_dim } }
1321         \dim_gzero:N \g_@@_blocks_wd_dim
1322     }
1323     \dim_compare:nNt { \g_@@_blocks_dp_dim } > { \c_zero_dim }
1324     {
1325         \box_set_dp:Nn \l_@@_cell_box
1326         { \dim_max:nn { \box_dp:N \l_@@_cell_box } { \g_@@_blocks_dp_dim } }
1327         \dim_gzero:N \g_@@_blocks_dp_dim
1328     }
1329     \dim_compare:nNt { \g_@@_blocks_ht_dim } > { \c_zero_dim }
1330     {
1331         \box_set_ht:Nn \l_@@_cell_box
1332         { \dim_max:nn { \box_ht:N \l_@@_cell_box } { \g_@@_blocks_ht_dim } }
1333         \dim_gzero:N \g_@@_blocks_ht_dim
1334     }
1335 }
1336 \cs_new_protected:Npn \@@_cell_end:
1337 {

```

The following command is nullified in the tabulars.

```

1338     \@@_tuning_not_tabular_end:
1339     \hbox_set_end:
1340     \@@_cell_end_i:
1341 }
1342 \cs_new_protected:Npn \@@_cell_end_i:
1343 {

```

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.

```

1344     \g_@@_cell_after_hook_tl
1345     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
1346     \@@_adjust_size_box:
1347     \box_set_ht:Nn \l_@@_cell_box
1348     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1349     \box_set_dp:Nn \l_@@_cell_box
1350     { \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”).

```

1351     \@@_update_max_cell_width:

```

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

```

1352     \@@_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.

```

1353   \bool_if:NTF \g_@@_empty_cell_bool
1354     { \box_use_drop:N \l_@@_cell_box }
1355     {
1356       \bool_if:NTF \g_@@_not_empty_cell_bool
1357         { \@@_print_node_cell: }
1358         {
1359           \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
1360             { \@@_print_node_cell: }
1361             { \box_use_drop:N \l_@@_cell_box }
1362         }
1363     }
1364   \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
1365     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1366   \bool_gset_false:N \g_@@_empty_cell_bool
1367   \bool_gset_false:N \g_@@_not_empty_cell_bool
1368 }

```

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

```

1369 \cs_new_protected:Npn \@@_update_max_cell_width:
1370 {
1371   \dim_gset:Nn \g_@@_max_cell_width_dim
1372     { \dim_max:nn { \g_@@_max_cell_width_dim } { \box_wd:N \l_@@_cell_box } }
1373 }

```

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`).

```

1374 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1375 {
1376   \@@_math_toggle:
1377   \hbox_set_end:
1378   \bool_if:NF \g_@@_rotate_bool
1379     {
1380       \hbox_set:Nn \l_@@_cell_box
1381         {
1382           \makebox [ \l_@@_col_width_dim ] [ s ]
1383             { \hbox_unpack_drop:N \l_@@_cell_box }
1384         }
1385     }
1386   \@@_cell_end_i:
1387 }

```

```

1388 \pgfset
1389 {
1390   nicematrix / cell-node /.style =
1391   {
1392     inner-sep = \c_zero_dim ,
1393     minimum-width = \c_zero_dim
1394   }
1395 }

```

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.

```

1396 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1397 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1398 {
1399   \use:c
1400   {
1401     __siunitx_table_align_
1402     \bool_if:NTF \l__siunitx_table_text_bool
1403       { \l__siunitx_table_align_text_tl }
1404       { \l__siunitx_table_align_number_tl }
1405     :n
1406   }
1407   { #1 }
1408 }

```

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!

```

1409 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1410 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1411 {
1412   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1413   \hbox:n
1414   {
1415     \pgfsys@markposition
1416     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1417   }
1418   #1
1419   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1420   \hbox:n
1421   {
1422     \pgfsys@markposition
1423     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1424   }
1425 }

1426 \cs_new_protected:Npn \@@_print_node_cell:
1427 {
1428   \socket_use:nn { nicematrix / siunitx-wrap }
1429   { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1430 }

```

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

```

1431 \cs_new_protected:Npn \@@_node_cell:
1432 {
1433   \pgfpicture
1434   \pgfsetbaseline \c_zero_dim

```

```

1435 \pgfrememberpicturepositiononpagetrue
1436 \pgfset { nicematrix / cell-node }
1437 \pgfnode
1438 { rectangle }
1439 { base }
1440 {

```

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).

```

1441 \sys_if_engine_xetex:T { \set@color }
1442 \box_use:N \l_@@_cell_box
1443 }
1444 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1445 { \l_@@_pgf_node_code_tl }
1446 \str_if_empty:NF \l_@@_name_str
1447 {
1448 \pgfnodealias
1449 { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1450 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1451 }
1452 \endpgfpicture
1453 }

```

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).

```

1454 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1455 {
1456 \bool_if:nTF { #1 } { \tl_gput_left:ce } { \tl_gput_right:ce }
1457 { g_@@_ #2 _ lines _ tl }
1458 {
1459 \use:c { @@ _ draw _ #2 : nnn }
1460 { \int_use:N \c@iRow }
1461 { \int_use:N \c@jCol }
1462 { \exp_not:n { #3 } }
1463 }
1464 }

1465 \cs_new_protected:Npn \@@_array:n
1466 {
1467 \dim_set:Nn \col@sep
1468 { \bool_if:nTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1469 \dim_compare:nNnTF { \l_@@_tabular_width_dim } = { \c_zero_dim }
1470 { \def \@halignto { } }
1471 { \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`.

```
1472 \tabarray
1473 \l_@@_baseline_tl may have the value t, c or b. However, if the value is b, we compose
1474 the \array (of array) with the option t and the right translation will be done further. Re-
1475 mark that \str_if_eq:eeTF is fully expandable and we need something fully expandable here.
1476 \str_if_eq:ee(TF) is faster than \str_if_eq:nn(TF).
```

```
1473 [ \str_if_eq:eeTF { \l_@@_baseline_tl } { c } { c } { t } ]
1474 }
1475 \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`.

```
1476 \bool_if:NTF \c_@@_revtex_bool
1477 { \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`.

```
1478 { \cs_set_eq:cN { \@@_old_ar@ialign: } \ar@ialign }
```

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

```
1479 \cs_new_protected:Npn \@@_create_row_node:
1480 {
1481   \int_compare:nNt { \c@iRow } > { \g_@@_last_row_node_int }
1482   {
1483     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1484     \@@_create_row_node_i:
1485   }
1486 }
1487 \cs_new_protected:Npn \@@_create_row_node_i:
1488 {
```

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

```
1489 \hbox
1490 {
1491   \bool_if:NT \l_@@_code_before_bool
1492   {
1493     \vtop
1494     {
1495       \skip_vertical:N 0.5\arrayrulewidth
1496       \pgfsys@markposition
1497       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1498       \skip_vertical:N -0.5\arrayrulewidth
1499     }
1500   }
1501   \pgfpicture
1502   \pgfrememberpicturerepositiononpagetrue
1503   \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1504   { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1505   \str_if_empty:NF \l_@@_name_str
1506   {
1507     \pgfnodealias
1508     { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1509     { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1510   }
1511   \endpgfpicture
1512 }
1513 }
```

```
1514 \cs_new_protected:Npn \@@_in_everycr:
1515 {
1516   \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
```

```

1517 \tbl_update_cell_data_for_next_row:
1518 \int_gzero:N \c_jCol
1519 \bool_gset_false:N \g_@@_after_col_zero_bool
1520 \bool_if:NF \g_@@_row_of_col_done_bool
1521 {
1522   \@@_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).

```

1523 \clist_if_empty:NF \l_@@_hlines_clist
1524 {
1525   \str_if_eq:eeF { \l_@@_hlines_clist } { all }
1526   {
1527     \clist_if_in:NtT
1528       \l_@@_hlines_clist
1529       { \int_eval:n { \c@iRow + 1 } }
1530   }
1531 {

```

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.

```

1532 \int_compare:nNnT { \c@iRow } > { -1 }
1533 {
1534   \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
1535   { \hrule height \arrayrulewidth width \c_zero_dim }
1536 }
1537 }
1538 }
1539 }
1540 }

```

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

```

1541 \cs_set_protected:Npn \@@_renew_dots:
1542 {
1543   \cs_set_eq:NN \ldots \@@_Ldots:
1544   \cs_set_eq:NN \cdots \@@_Cdots:
1545   \cs_set_eq:NN \vdots \@@_Vdots:
1546   \cs_set_eq:NN \ddots \@@_Ddots:
1547   \cs_set_eq:NN \iddots \@@_Iddots:
1548   \cs_set_eq:NN \dots \@@_Ldots:
1549   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1550 }

```

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

```

1551 \hook_gput_code:nnn { begindocument } { . }
1552 {
1553   \IfPackageLoadedTF { booktabs }
1554   {
1555     \cs_new_protected:Npn \@@_patch_booktabs:
1556     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1557   }
1558   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1559 }

```

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

The box `\@arstrutbox` is a box constructed in the beginning of the environment `{array}`. The construction of that box takes into account the current value of `\arraystretch`⁶ and `\extrarowheight` (of `array`). That box is inserted (via `\@arstrut`) in the beginning of each row of the array. That’s why we use the dimensions of that box to initialize the variables which will be the dimensions of the potential first and last row of the environment. This initialization must be done after the creation of `\@arstrutbox` and that’s why we do it in the `\ialign`.

```

1560 \cs_new_protected:Npn \@@_some_initialization:
1561 {
1562   \@@_everycr:
1563   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1564   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1565   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1566   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1567   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1568   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1569 }

```

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

```

1570 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1571 {

```

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

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

```

1572   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1573   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1574   \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.

```

1575   \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1576   \fp_gzero:N \g_@@_total_X_weight_fp
1577   \bool_gset_false:N \g_@@_V_of_X_bool
1578   \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1579   \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol
1580   \@@_patch_booktabs:
1581   \box_clear_new:N \l_@@_cell_box
1582   \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}`).

```

1583   \bool_if:NT \l_@@_small_bool
1584   {

```

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

```

1585     \def \arraystretch { 0.47 }
1586     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1587     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1588 }

```

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!

```

1589     \bool_if:NT \g_@@_create_cell_nodes_bool
1590     {
1591         \tl_put_right:Nn \@@_begin_of_row:
1592         {
1593             \pgfsys@markposition
1594             { \@@_env: - row - \int_use:N \c@iRow - base }
1595         }
1596         \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1597     }

```

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

```

1598     \bool_if:NF \c_@@_revtex_bool
1599     {
1600         \def \ar@ialign
1601         {
1602             \tbl_init_cell_data_for_table:
1603             \@@_some_initialization:
1604             \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`.

```

1605             \cs_set_eq:Nc \ar@ialign { @@_old_ar@ialign: }
1606             \halign
1607         }
1608     }

```

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).

```

1609     \bool_if:NT \c_@@_revtex_bool
1610     {
1611         \IfPackageLoadedT { colortbl }
1612         { \cs_set_protected:Npn \CT@setup { } }
1613     }

```

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).

```

1614     \cs_set_eq:NN \@@_old_ldots: \ldots
1615     \cs_set_eq:NN \@@_old_cdots: \cdots
1616     \cs_set_eq:NN \@@_old_vdots: \vdots
1617     \cs_set_eq:NN \@@_old_ddots: \ddots
1618     \cs_set_eq:NN \@@_old_iddots: \iddots
1619     \bool_if:NTF \l_@@_standard_cline_bool
1620     { \cs_set_eq:NN \cline \@@_standard_cline: }
1621     { \cs_set_eq:NN \cline \@@_cline: }
1622     \cs_set_eq:NN \Ldots \@@_Ldots:

```

```

1623 \cs_set_eq:NN \Cdots \@@_Cdots:
1624 \cs_set_eq:NN \Vdots \@@_Vdots:
1625 \cs_set_eq:NN \Ddots \@@_Ddots:
1626 \cs_set_eq:NN \Iddots \@@_Iddots:
1627 \cs_set_eq:NN \Hline \@@_Hline:
1628 \cs_set_eq:NN \Hspace \@@_Hspace:
1629 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1630 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1631 \cs_set_eq:NN \Block \@@_Block:
1632 \cs_set_eq:NN \rotate \@@_rotate:
1633 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1634 \cs_set_eq:NN \dotfill \@@_dotfill:
1635 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1636 \cs_if_free:NT \Body { \cs_set_eq:NN \Body \@@_Body: }
1637 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1638 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1639 \cs_set_eq:NN \TopRule \@@_TopRule
1640 \cs_set_eq:NN \MidRule \@@_MidRule
1641 \cs_set_eq:NN \BottomRule \@@_BottomRule
1642 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1643 \cs_set_eq:NN \Hbrace \@@_Hbrace
1644 \cs_set_eq:NN \Vbrace \@@_Vbrace
1645 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1646   { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1647 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1648 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1649 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1650 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1651 \int_compare:nNnT { \l_@@_first_row_int } > { \c_zero_int }
1652   { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1653 \int_compare:nNnT { \l_@@_last_row_int } < { \c_zero_int }
1654   { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1655 \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:`.

```

1656 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1657 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1658   { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1659 \@@_revert_colortbl:

```

If there is one or several commands `\tablarnote` 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!).

```

1660 \tl_if_exist:NT \l_@@_note_in_caption_tl
1661 {
1662   \tl_if_empty:NF \l_@@_note_in_caption_tl
1663   {
1664     \int_gset:Nn \g_@@_notes_caption_int { \l_@@_note_in_caption_tl }
1665     \int_gset:Nn \c@tablarnote { \l_@@_note_in_caption_tl }
1666   }
1667 }

```

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).

```

1668 \seq_gclear:N \g_@@_multicolumn_cells_seq
1669 \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).

```
1670 \int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }
```

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

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

```
1671 \int_gzero:N \g_@@_row_total_int
```

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

```
1672 \int_gzero:N \g_@@_col_total_int
1673 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1674 \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.

```
1675 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1676 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1677 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1678 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1679 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1680 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1681 \tl_gclear:N \g_nicematrix_code_before_tl
1682 \tl_gclear:N \g_@@_pre_code_before_tl
```

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

```
1683 \dim_zero_new:N \l_@@_left_delim_dim
1684 \dim_zero_new:N \l_@@_right_delim_dim
1685 \bool_if:NTF \g_@@_delims_bool
1686 {
```

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

```
1687 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1688 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1689 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1690 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1691 }
1692 {
1693 \dim_gset:Nn \l_@@_left_delim_dim
1694 { 2 \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1695 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1696 }
1697 }
```

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

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

```
1698 \cs_new_protected:Npn \@@_pre_array:
1699 {
1700 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1701 \int_gzero_new:N \c@iRow
1702 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1703 \int_gzero_new:N \c@jCol
```

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

```

1704 \int_compare:nNnT \l_@@_last_row_int > 0
1705   { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1706 \int_compare:nNnT \l_@@_last_col_int > 0
1707   { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1708 \bool_if:NT \g_@@_aux_found_bool
1709   {
1710     \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1711     \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1712     \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1713     \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1714   }

```

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

```

1715 \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
1716   {
1717     \bool_set_true:N \l_@@_last_row_without_value_bool
1718     \bool_if:NT \g_@@_aux_found_bool
1719       { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1720   }
1721 \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
1722   {
1723     \bool_if:NT \g_@@_aux_found_bool
1724       { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1725   }

```

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”.

```

1726 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
1727   {
1728     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1729       {
1730         \dim_compare:nNnT { \g_@@_ht_last_row_dim } < { \box_ht:N \l_@@_cell_box }
1731           { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1732         \dim_compare:nNnT { \g_@@_dp_last_row_dim } < { \box_dp:N \l_@@_cell_box }
1733           { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1734       }
1735   }

1736 \seq_gclear:N \g_@@_cols_vlism_seq
1737 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1738 \bool_if:NT \l_@@_code_before_bool { \@@_exec_code_before: }

```

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

```

1739 \@@_pre_array_after_CodeBefore:

```

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

```

1740 \hbox_set:Nw \l_@@_the_array_box
1741 \skip_horizontal:N \l_@@_left_margin_dim
1742 \skip_horizontal:N \l_@@_extra_left_margin_dim
1743 \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).

```

1744 \m@th
1745 $ % $
1746 \bool_if:NTF \l_@@_light_syntax_bool
1747 { \use:c { @@-light-syntax } }
1748 { \use:c { @@-normal-syntax } }
1749 }

```

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

```

1750 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1751 {
1752   \tl_set:Nn \l_tmpa_tl { #1 }
1753   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1754   { \@@_rescan_for_spanish:N \l_tmpa_tl }
1755   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1756   \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...

```

1757 \@@_pre_array:
1758 }

```

9 The `\CodeBefore`

```

1759 \cs_new_protected_nopar:Npn \@@_Body: { \@@_fatal:n { Body~alone } }

```

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

```

1760 \cs_new_protected:Npn \@@_pre_code_before:
1761 {

```

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.

```

1762 \pgfsys@markposition { \@@_env: - position }
1763 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1764 \pgfpicture
1765 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1766 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int + 1 }
1767 {
1768   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1769   \pgfcoordinate { \@@_env: - row - ##1 }
1770   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1771 }

```

Now, the recreation of the `col` nodes.

```

1772 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int + 1 }
1773 {
1774   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:

```

```

1775     \pgfcoordinate { \@@_env: - col - ##1 }
1776     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1777 }

```

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

```

1778 \bool_if:NT \g_@@_create_cell_nodes_bool { \@@_recreate_cell_nodes: }
1779 \endpgfpicture

```

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

```

1780 \@@_create_diag_nodes:

```

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

```

1781 \@@_create_blocks_nodes:
1782 \IfPackageLoadedT { tikz }
1783 {
1784     \tikzset
1785     {
1786         every-picture / .style =
1787         { overlay , name~prefix = \@@_env: - }
1788     }
1789 }
1790 \cs_set_eq:NN \cellcolor \@@_cellcolor
1791 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1792 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1793 \cs_set_eq:NN \rowcolor \@@_rowcolor
1794 \cs_set_eq:NN \rowcolors \@@_rowcolors
1795 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1796 \cs_set_eq:NN \arraycolor \@@_arraycolor
1797 \cs_set_eq:NN \columncolor \@@_columncolor
1798 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1799 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1800 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1801 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1802 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1803 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1804 }

```

```

1805 \cs_new_protected:Npn \@@_exec_code_before:
1806 {

```

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...

```

1807 \clist_map_inline:Nn \l_@@_corners_cells_clist
1808 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1809 \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`.

```

1810 \@@_add_to_colors_seq:nn { { nocolor } } { }
1811 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1812 \group_begin:

```

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

```

1813 \if_mode_math:
1814     \@@_exec_code_before_i:
1815 \else:
1816     $ % $
1817     \@@_exec_code_before_i:
1818     $ % $
1819 \fi:
1820 \group_end:
1821 }

```

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`).

```

1822 \cs_new_protected:Npn \@@_exec_code_before_i:
1823 {
1824   \int_compare:nNt { \char_value_catcode:n { 60 } } = { 13 }
1825   { \@@_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`.

```

1826   \exp_last_unbraced:No \@@_CodeBefore_keys:
1827   \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.

```

1828     \@@_actually_color:
1829     \l_@@_code_before_tl
1830     \q_stop
1831 }

```

```

1832 \keys_define:nn { nicematrix / CodeBefore }
1833 {
1834   create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1835   create-cell-nodes .default:n = true ,
1836   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1837   sub-matrix .value_required:n = true ,
1838   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1839   delimiters / color .value_required:n = true ,
1840   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1841 }

1842 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1843 {
1844   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1845   \@@_CodeBefore:w
1846 }

```

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.

```

1847 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1848 {
1849   \bool_if:NTF \g_@@_aux_found_bool
1850   {
1851     \@@_pre_code_before:
1852     \legacy_if:nF { measuring@ } { #1 }
1853   }

```

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

```

1854   {
1855     \pgfsys@markposition { \@@_env: - position }
1856     \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1857     \pgfpicture
1858     \pgf@relevantforpicturesizefalse
1859     \endpgfpicture

```

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

```

1860     \pgfpicture
1861     \pgfrememberpicturepositiononpagetrue
1862     \endpgfpicture

```

The following picture corresponds to \@@_create_blocks_nodes:.

```

1863 \pgfpicture
1864 \pgf@relevantforpicturesizefalse
1865 \pgfrememberpicturepositiononpagetrue
1866 \endpgfpicture

```

The following picture corresponds \@@_actually_color:

```

1867 \pgfpicture
1868 \pgf@relevantforpicturesizefalse
1869 \endpgfpicture
1870 }
1871 }

```

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.

```

1872 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1873 {
1874   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
1875   {
1876     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1877     \pgfcoordinate { \@@_env: - row - ##1 - base }
1878     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1879     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
1880     {
1881       \cs_if_exist:cT
1882       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1883       {
1884         \pgfsys@getposition
1885         { \@@_env: - ##1 - #####1 - NW }
1886         \@@_node_position:
1887         \pgfsys@getposition
1888         { \@@_env: - ##1 - #####1 - SE }
1889         \@@_node_position_i:
1890         \@@_pgf_rect_node:nnn
1891         { \@@_env: - ##1 - #####1 }
1892         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1893         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1894       }
1895     }
1896   }
1897   \@@_create_extra_nodes:
1898   \@@_create_aliases_last:
1899 }

1900 \cs_new_protected:Npn \@@_create_aliases_last:
1901 {
1902   \int_step_inline:nn { \c@iRow }
1903   {
1904     \pgfnodealias
1905     { \@@_env: - ##1 - last }
1906     { \@@_env: - ##1 - \int_use:N \c@jCol }
1907   }
1908   \int_step_inline:nn { \c@jCol }
1909   {
1910     \pgfnodealias
1911     { \@@_env: - last - ##1 }
1912     { \@@_env: - \int_use:N \c@iRow - ##1 }
1913   }
1914   \pgfnodealias
1915   { \@@_env: - last - last }
1916   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }

```

```

1917 }

1918 \cs_new_protected:Npn \@@_create_blocks_nodes:
1919 {
1920   \pgfpicture
1921   \pgf@relevantforpicturesizefalse
1922   \pgfrememberpicturepositiononpagetrue
1923   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1924     { \@@_create_one_block_node:nnnnn ##1 }
1925   \endpgfpicture
1926 }

```

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

```

1927 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1928 {
1929   \tl_if_empty:nF { #5 }
1930   {
1931     \@@_qpoint:n { col - #2 }
1932     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1933     \@@_qpoint:n { #1 }
1934     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1935     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1936     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1937     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1938     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1939     \@@_pgf_rect_node:nnnnn
1940       { \@@_env: - #5 }
1941       { \dim_use:N \l_tmpa_dim }
1942       { \dim_use:N \l_tmpb_dim }
1943       { \dim_use:N \l_@@_tmpc_dim }
1944       { \dim_use:N \l_@@_tmpd_dim }
1945   }
1946 }

1947 \cs_new_protected:Npn \@@_patch_for_revtext:
1948 {
1949   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1950   \cs_set_eq:NN \@array \@array@array
1951   \cs_set_eq:NN \@tabular \@tabular@array
1952   \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1953   \cs_set_eq:NN \array \array@array
1954   \cs_set_eq:NN \endarray \endarray@array
1955   \cs_set:Npn \endtabular { \endarray $\egroup } % $
1956   \cs_set_eq:NN \@mkpream \@mkpream@array
1957   \cs_set_eq:NN \@classx \@classx@array
1958   \cs_set_eq:NN \insert@column \insert@column@array
1959   \cs_set_eq:NN \@arraycr \@arraycr@array
1960   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1961   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1962 }

```

10 The environment `{NiceArrayWithDelims}`

```

1963 \NewDocumentEnvironment { NiceArrayWithDelims }
1964 { m m O { } m ! O { } t \CodeBefore }
1965 {

```

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).

```

1966 \bool_if:NT \c_@@_revtex_bool { \@@_patch_for_revtex: }
1967 \@@_provide_pgfsyspdfmark:
1968 \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.

```

1969 \bgroup

1970 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1971 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1972 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1973 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

1974 \int_gzero:N \g_@@_block_box_int
1975 \dim_gzero:N \g_@@_width_last_col_dim
1976 \dim_gzero:N \g_@@_width_first_col_dim
1977 \bool_gset_false:N \g_@@_row_of_col_done_bool
1978 \str_if_empty:NT \g_@@_name_env_str
1979 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1980 \bool_if:NTF \l_@@_tabular_bool
1981 { \mode_leave_vertical: }
1982 { \@@_test_if_math_mode: }
1983 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1984 \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.

```

1985 \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.

```

1986 \cs_if_exist:NT \tikz@library@external@loaded
1987 {
1988   \tikzexternaldisable
1989   \cs_if_exist:NT \ifstandalone
1990     { \tikzset { external / optimize = false } }
1991 }

```

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

```

1992 \int_gincr:N \g_@@_env_int
1993 \bool_if:NF \l_@@_block_auto_columns_width_bool
1994 { \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.

```

1995 \seq_gclear:N \g_@@_blocks_seq
1996 \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`.

```

1997 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1998 \seq_gclear:N \g_@@_pos_of_xdots_seq
1999 \tl_gclear_new:N \g_@@_code_before_tl
2000 \tl_gclear:N \g_@@_row_style_tl

```

We load all the information written in the aux file during previous compilations corresponding to the current environment.

⁸e.g. `\color[rgb]{0.5,0.5,0}`


```

2001 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2002 {
2003   \bool_gset_true:N \g_@@_aux_found_bool
2004   \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2005 }
2006 { \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.

```

2007 \tl_gclear:N \g_@@_aux_tl
2008 \tl_if_empty:NF \g_@@_code_before_tl
2009 {
2010   \bool_set_true:N \l_@@_code_before_bool
2011   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
2012 }
2013 \tl_if_empty:NF \g_@@_pre_code_before_tl
2014 { \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.

```

2015 \bool_if:NTF \g_@@_delims_bool
2016 { \keys_set:nn { nicematrix / pNiceArray } }
2017 { \keys_set:nn { nicematrix / NiceArray } }
2018 { #3 , #5 }

2019 \@@_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:.

```

2020 \bool_if:nTF { #6 } { \@@_CodeBefore_Body:w } { \@@_pre_array: }
2021 }

```

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

```

2022 {
2023   \bool_if:NTF \l_@@_light_syntax_bool
2024   { \use:c { end @@-light-syntax } }
2025   { \use:c { end @@-normal-syntax } }
2026   $ % $
2027   \skip_horizontal:N \l_@@_right_margin_dim
2028   \skip_horizontal:N \l_@@_extra_right_margin_dim
2029   \hbox_set_end:
2030   \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.

```

2031 \bool_if:NT \l_@@_width_used_bool
2032 {
2033   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2034   { \@@_error_or_warning:n { width~without~X~columns } }
2035 }

```

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

```

2036 \fp_compare:nNnT { \g_@@_total_X_weight_fp } > { \c_zero_fp }
2037 { \@@_compute_width_X: }

```

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

```

2038 \int_compare:nNt { \l_@@_last_row_int } > { -2 }
2039 {
2040   \bool_if:NF \l_@@_last_row_without_value_bool
2041   {
2042     \int_compare:nNf { \l_@@_last_row_int } = { \c@iRow }
2043     {
2044       \@@_error:n { Wrong~last~row }
2045       \int_set_eq:NN \l_@@_last_row_int \c@iRow
2046     }
2047   }
2048 }

```

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

```

2049 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2050 \bool_if:NTF \g_@@_last_col_found_bool
2051 { \int_gdecr:N \c@jCol }
2052 {
2053   \int_compare:nNt { \l_@@_last_col_int } > { -1 }
2054   { \@@_error:n { last~col~not~used } }
2055 }

```

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

```

2056 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2057 \int_compare:nNt { \l_@@_last_row_int } > { -1 }
2058 { \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. 93).

```

2059 \int_if_zero:nT { \l_@@_first_col_int }
2060 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2061 \bool_if:nTF { ! \g_@@_delims_bool }
2062 {
2063   \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
2064   { \@@_use_arraybox_with_notes_c: }
2065   {
2066     \str_if_eq:eeTF { \l_@@_baseline_tl } { b }
2067     { \@@_use_arraybox_with_notes_b: }
2068     { \@@_use_arraybox_with_notes: }
2069   }
2070 }

```

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).

```

2071 {
2072   \int_if_zero:nTF { \l_@@_first_row_int }
2073   {
2074     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2075     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2076   }
2077   { \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

```

2078 \int_compare:nNtTF { \l_@@_last_row_int } > { -2 }

```

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

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

```

2079     {
2080       \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2081       \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2082     }
2083     { \dim_zero:N \l_tmpb_dim }
2084   \hbox_set:Nn \l_tmpa_box
2085   {
2086     \m@th
2087     $ % $
2088     \@@_color:o \l_@@_delimiters_color_tl
2089     \exp_after:wN \left \g_@@_left_delim_tl
2090     \vcenter
2091     {

```

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.

```

2092       \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2093       \hbox
2094       {
2095         \bool_if:NTF \l_@@_tabular_bool
2096         { \skip_horizontal:n { - \tabcolsep } }
2097         { \skip_horizontal:n { - \arraycolsep } }
2098         \@@_use_arraybox_with_notes_c:
2099         \bool_if:NTF \l_@@_tabular_bool
2100         { \skip_horizontal:n { - \tabcolsep } }
2101         { \skip_horizontal:n { - \arraycolsep } }
2102       }

```

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

```

2103       \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2104     }
2105     \exp_after:wN \right \g_@@_right_delim_tl
2106     $ % $
2107   }

```

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.

```

2108     \bool_if:NTF \l_@@_delimiters_max_width_bool
2109     {
2110       \@@_put_box_in_flow_bis:nn
2111       { \g_@@_left_delim_tl }
2112       { \g_@@_right_delim_tl }
2113     }
2114     \@@_put_box_in_flow:
2115   }

```

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. 94).

```

2116     \bool_if:NT \g_@@_last_col_found_bool
2117     { \skip_horizontal:N \g_@@_width_last_col_dim }
2118     \bool_if:NT \l_@@_preamble_bool
2119     {
2120       \int_compare:nNnT { \c@jCol } < { \g_@@_static_num_of_col_int }
2121       { \@@_err_columns_not_used: }
2122     }
2123     \@@_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.

```

2124     \egroup

```

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

```

2125     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2126     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }

```

```

2127 \iow_now:Ne \@mainaux
2128 {
2129     \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2130     \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2131     { \exp_not:o \g_@@_aux_tl }
2132 }
2133 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2134 \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2135 }

```

This is the end of the environment {NiceArrayWithDelims}.

```

2136 \cs_new_protected:Npn \@_err_columns_not_used:
2137 {
2138     \@_warning:n { columns~not~used }
2139     \cs_gset:Npn \@_err_columns_not_used: { }
2140 }

```

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 .

```

2141 \cs_new_protected:Npn \@_compute_width_X:
2142 {
2143     \tl_gput_right:Ne \g_@@_aux_tl
2144     {
2145         \bool_set_true:N \l_@@_X_columns_aux_bool
2146         \dim_set:Nn \l_@@_X_columns_dim
2147     }

```

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`

```

2148     \bool_lazy_and:nnTF
2149     { \g_@@_V_of_X_bool }
2150     { \l_@@_X_columns_aux_bool }
2151     { \dim_use:N \l_@@_X_columns_dim }
2152     {
2153         \dim_compare:nNnTF
2154         {
2155             \dim_abs:n
2156             { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2157         }
2158         <
2159         { 0.001 pt }
2160         { \dim_use:N \l_@@_X_columns_dim }
2161         {
2162             \dim_eval:n
2163             {
2164                 \l_@@_X_columns_dim
2165                 +
2166                 \fp_to_dim:n
2167                 {
2168                     (
2169                         \dim_eval:n
2170                         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2171                     )
2172                     / \fp_use:N \g_@@_total_X_weight_fp
2173                 }
2174             }
2175         }
2176     }

```

```

2177     }
2178   }
2179 }

```

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`.

```

2180 \cs_new_protected:Npn \@@_transform_preamble:
2181 {
2182   \@@_transform_preamble_i:
2183   \@@_transform_preamble_ii:
2184 }
2185 \cs_new_protected:Npn \@@_transform_preamble_i:
2186 {
2187   \int_gzero:N \c@jCol

```

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

```

2188   \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.

```

2189   \bool_gset_false:N \g_tmpb_bool

```

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

```

2190   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2191   \int_zero:N \l_tmpa_int
2192   \tl_gclear:N \g_@@_array_preamble_tl
2193   \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2194   {
2195     \tl_gset:Nn \g_@@_array_preamble_tl
2196     { ! { \skip_horizontal:N \arrayrulewidth } }
2197   }
2198   {
2199     \clist_if_in:NnT \l_@@_vlines_clist 1
2200     {
2201       \tl_gset:Nn \g_@@_array_preamble_tl
2202       { ! { \skip_horizontal:N \arrayrulewidth } }
2203     }
2204   }

```

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

```

2205   \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2206   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2207   \@@_replace_columncolor:
2208 }

2209 \cs_new_protected:Npn \@@_transform_preamble_ii:
2210 {

```

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

```

2211 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2212 {
2213   \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2214   { \bool_gset_true:N \g_@@_delims_bool }
2215 }
2216 { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2217 \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).

```

2218 \int_if_zero:nTF { \l_@@_first_col_int }
2219 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2220 {
2221   \bool_if:NF \g_@@_delims_bool
2222   {
2223     \bool_if:NF \l_@@_tabular_bool
2224     {
2225       \clist_if_empty:NT \l_@@_vlines_clist
2226       {
2227         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2228         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2229       }
2230     }
2231   }
2232 }
2233 \int_compare:nNnTF { \l_@@_last_col_int } > { -1 }
2234 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2235 {
2236   \bool_if:NF \g_@@_delims_bool
2237   {
2238     \bool_if:NF \l_@@_tabular_bool
2239     {
2240       \clist_if_empty:NT \l_@@_vlines_clist
2241       {
2242         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2243         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2244       }
2245     }
2246   }
2247 }

```

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

```

2248 \tag_if_active:F
2249 {

```

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

```

2250 \dim_compare:nNnT { \l_@@_tabular_width_dim } = { \c_zero_dim }
2251 {
2252   \tl_gput_right:Nn \g_@@_array_preamble_tl
2253   { > { \@@_err_too_many_cols: } 1 }
2254 }
2255 }
2256 }

```

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

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

```
2257 \cs_new_protected:Npn \@@_rec_preamble:n #1
2258 {
```

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

```
2259 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2260 { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2261 {
```

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

```
2262 \cs_if_exist:cTF { NC @ find @ #1 }
2263 {
2264 \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2265 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2266 }
2267 {
2268 \str_if_eq:nnTF { #1 } { S }
2269 { \@@_fatal:n { unknown~column~type~S } }
2270 { \@@_fatal:nn { unknown~column~type } { #1 } }
2271 }
2272 }
2273 }
```

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

```
2274 \cs_new_protected:Npn \@@_c: #1
2275 {
2276 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2277 \tl_gclear:N \g_@@_pre_cell_tl
2278 \tl_gput_right:Nn \g_@@_array_preamble_tl
2279 { > \@@_cell_begin: c < \@@_cell_end: }
```

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

```
2280 \int_gincr:N \c@jCol
2281 \@@_rec_preamble_after_col:n
2282 }

2283 \cs_new_protected:Npn \@@_l: #1
2284 {
2285 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2286 \tl_gclear:N \g_@@_pre_cell_tl
2287 \tl_gput_right:Nn \g_@@_array_preamble_tl
2288 {
2289 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2290 l
2291 < \@@_cell_end:
2292 }
2293 \int_gincr:N \c@jCol
2294 \@@_rec_preamble_after_col:n
2295 }
```

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`.

```

2296 \cs_new_protected:Npn \@@_r: #1
2297 {
2298   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2299   \tl_gclear:N \g_@@_pre_cell_tl
2300   \tl_gput_right:Nn \g_@@_array_preamble_tl
2301   {
2302     > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2303     r
2304     < \@@_cell_end:
2305   }
2306   \int_gincr:N \c@jCol
2307   \@@_rec_preamble_after_col:n
2308 }

```

For ! and @

```

2309 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2310 {
2311   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2312   \@@_rec_preamble:n
2313 }
2314 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2315 \cs_new_protected:cpn { @@ _ | : } #1
2316 {
\l_tmpa_int is the number of successive occurrences of |
2317   \int_incr:N \l_tmpa_int
2318   \@@_make_preamble_i_i:n
2319 }
2320 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2321 {

```

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

```

2322   \str_if_eq:nnTF { #1 } { | }
2323   { \use:c { @@ _ | : } | }
2324   { \@@_make_preamble_i_ii:nn { } #1 }
2325 }

```

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

```

2326 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2327 {
2328   \str_if_eq:nnTF { #2 } { [ ]
2329   { \@@_make_preamble_i_ii:nw { #1 } [ ]
2330   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2331   }
2332 \cs_new_protected:Npn \@@_make_preamble_i_iii:nw #1 [ #2 ]
2333 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2334 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2335 {
2336   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2337   \tl_gput_right:Ne \g_@@_array_preamble_tl
2338   {

```

Here, the command \dim_use:N is mandatory.

```

2339   \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2340   }
2341   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2342   {
2343     \@@_vline:n
2344     {
2345       position = \int_eval:n { \c@jCol + 1 } ,

```



```

2346     multiplicity = \int_use:N \l_tmpa_int ,
2347     total-width = \dim_use:N \l_@@_rule_width_dim ,
2348     #2
2349 }

```

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.

```

2350     }
2351     \int_zero:N \l_tmpa_int
2352     \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2353     \@@_rec_preamble:n #1
2354 }

2355 \cs_new_protected:cpn { @@_ > : } #1 #2
2356 {
2357     \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2358     \@@_rec_preamble:n
2359 }

2360 \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.

```

2361 \keys_define:nn { nicematrix / p-column }
2362 {
2363     r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2364     r .value_forbidden:n = true ,
2365     c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2366     c .value_forbidden:n = true ,
2367     l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2368     l .value_forbidden:n = true ,
2369     S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2370     S .value_forbidden:n = true ,
2371     p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2372     p .value_forbidden:n = true ,
2373     t .meta:n = p ,
2374     m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2375     m .value_forbidden:n = true ,
2376     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2377     b .value_forbidden:n = true
2378 }

```

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

```

2379 \cs_new_protected:Npn \@@_p: #1
2380 {
2381     \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2382     \@@_make_preamble_ii_i:n
2383 }
2384 \cs_set_eq:NN \@@_b: \@@_p:
2385 \cs_set_eq:NN \@@_m: \@@_p:

2386 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2387 {
2388     \str_if_eq:nnTF { #1 } { [ ]
2389         { \@@_make_preamble_ii_ii:w [ ]
2390             { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2391         }
2392     }
2393     \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2394     { \@@_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.

```
2394 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2395 {
```

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).

```
2396 \str_set:Nn \l_@@_hpos_col_str { j }
2397 \@@_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`.

```
2398 \setlength { \l_tmpa_dim } { #2 }
2399 \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2400 }
2401 \cs_new_protected:Npn \@@_keys_p_column:n #1
2402 { \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.

```
2403 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2404 {
```

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

```
2405 \use:e
2406 {
2407 \@@_make_preamble_ii_vi:nnnnnnnn
2408 { \str_if_eq:eeTF { \l_@@_vpos_col_str } { p } { t } { b } }
2409 { #1 }
2410 {
```

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.

```
2411 \str_if_eq:eeTF { \l_@@_hpos_col_str } { j }
2412 { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2413 {
```

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

```
2414 \def \exp_not:N \l_@@_hpos_cell_tl
2415 { \str_lowercase:f { \l_@@_hpos_col_str } }
2416 }
2417 \IfPackageLoadedTF { ragged2e }
2418 {
2419 \str_case:on \l_@@_hpos_col_str
2420 {
```

The following `\exp_not:N` are mandatory.

```
2421 c { \exp_not:N \Centering }
2422 l { \exp_not:N \RaggedRight }
2423 r { \exp_not:N \RaggedLeft }
2424 }
2425 }
2426 {
2427 \str_case:on \l_@@_hpos_col_str
2428 {
2429 c { \exp_not:N \centering }
2430 l { \exp_not:N \raggedright }
2431 r { \exp_not:N \raggedleft }
2432 }
```

```

2433     }
2434     #3
2435 }
2436 { \str_if_eq:eeT { \l_@@_vpos_col_str } { m } \@@_center_cell_box: }
2437 { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_begin:w }
2438 { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_end: }
2439 { #2 }
2440 {
2441     \str_case:onF \l_@@_hpos_col_str
2442     {
2443         { j } { c }
2444         { si } { c }
2445     }

```

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

```

2446     { \str_lowercase:f \l_@@_hpos_col_str }
2447 }
2448 }

```

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

```

2449     \int_gincr:N \c@jCol
2450     \@@_rec_preamble_after_col:n
2451 }

```

#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*.

```

2452 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2453 {
2454     \str_if_eq:eeTF { \l_@@_hpos_col_str } { si }
2455     {
2456         \tl_gput_right:Nn \g_@@_array_preamble_tl
2457         { > \@@_test_if_empty_for_S: }
2458     }
2459     {
2460         \str_if_eq:eeTF { #7 } { varwidth }
2461         {
2462             \tl_gput_right:Nn \g_@@_array_preamble_tl
2463             { > \@@_test_if_empty_varwidth: }
2464         }
2465         { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2466     }
2467     \tl_gput_right:Nn \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2468     \tl_gclear:N \g_@@_pre_cell_tl
2469     \tl_gput_right:Nn \g_@@_array_preamble_tl
2470     {
2471         > {

```

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.

```

2472     \dim_set:Nn \l_@@_col_width_dim { #2 }
2473     \@@_cell_begin:

```

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

```
2474 \use:c { #7 } [ #1 ] { #2 }
```

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

```
2475 \everypar
2476 {
2477   \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2478   \everypar { }
2479 }
```

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

```
2480 #3
```

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

```
2481 \g_@@_row_style_tl
2482 \arraybackslash
2483 #5
2484 }
2485 #8
2486 < {
2487   #6
```

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

```
2488 \@finalstrut \@arstrutbox
2489 \use:c { end #7 }
```

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

```
2490 #4
2491 \@@_cell_end:
2492 }
2493 }
2494 }
```

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

```
2495 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2496 {
```

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 `&`).

```
2497 \group_align_safe_begin:
2498 \peek_meaning:NTF &
2499 { \@@_the_cell_is_empty: }
2500 {
2501   \peek_meaning:NTF \\\
2502   { \@@_the_cell_is_empty: }
2503   {
2504     \peek_meaning:NTF \crrc
2505     \@@_the_cell_is_empty:
2506     \group_align_safe_end:
2507   }
2508 }
2509 }
```

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

```
2510 \cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
2511 {
2512   \group_align_safe_begin:
2513   \peek_meaning:NTF &
2514   { \@@_the_cell_is_empty_varwidth: }
2515   {
```

```

2516     \peek_meaning:NTF \\\
2517     { \@@_the_cell_is_empty_varwidth: }
2518     {
2519         \peek_meaning:NTF \crcr
2520         \@@_the_cell_is_empty_varwidth:
2521         \group_align_safe_end:
2522     }
2523 }
2524 }
2525 \cs_new_protected:Npn \@@_the_cell_is_empty:
2526 {
2527     \group_align_safe_end:
2528     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2529     {

```

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

```

2530     \box_set_wd:Nn \l_@@_cell_box \c_zero_dim

```

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

```

2531     \skip_horizontal:N \l_@@_col_width_dim
2532 }
2533 }
2534 \cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
2535 {
2536     \group_align_safe_end:
2537     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2538     { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2539 }
2540 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2541 {
2542     \peek_meaning:NT \__siunitx_table_skip:n
2543     { \bool_gset_true:N \g_@@_empty_cell_bool }
2544 }

```

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.

```

2545 \cs_new_protected:Npn \@@_center_cell_box:
2546 {

```

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

```

2547     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2548     {
2549         \dim_compare:nNnT
2550         { \box_ht:N \l_@@_cell_box }
2551         >

```

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).

```

2552     { \box_ht:N \strutbox }
2553     {
2554         \hbox_set:Nn \l_@@_cell_box
2555         {
2556             \box_move_down:nn
2557             {
2558                 ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox

```

```

2559             + \baselineskip ) / 2
2560         }
2561         { \box_use:N \l_@@_cell_box }
2562     }
2563 }
2564 }
2565 }

```

For V (similar to the V of varwidth).

```

2566 \cs_new_protected:Npn \@@_V: #1 #2
2567 {
2568     \str_if_eq:nnTF { #2 } { [ ] }
2569     { \@@_make_preamble_V_i:w [ ] }
2570     { \@@_make_preamble_V_i:w [ ] { #2 } }
2571 }
2572 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2573 { \@@_make_preamble_V_ii:nn { #1 } }
2574 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2575 {
2576     \str_set:Nn \l_@@_vpos_col_str { p }
2577     \str_set:Nn \l_@@_hpos_col_str { j }
2578     \@@_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`.

```

2579     \setlength { \l_tmpa_dim } { #2 }
2580     \IfPackageLoadedTF { varwidth }
2581     { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2582     {
2583         \@@_error_or_warning:n { varwidth~not~loaded }
2584         \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2585     }
2586 }

```

For w and W

```

2587 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2588 \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.

```

2589 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2590 {
2591     \str_if_eq:nnTF { #3 } { s }
2592     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2593     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2594 }

```

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.

```

2595 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2596 {
2597     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2598     \tl_gclear:N \g_@@_pre_cell_tl
2599     \tl_gput_right:Nn \g_@@_array_preamble_tl
2600     {
2601         > {

```

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`.

```

2602         \setlength { \l_@@_col_width_dim } { #2 }
2603         \@@_cell_begin:
2604         \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2605     }
2606     c
2607     < {
2608         \@@_cell_end_for_w_s:
2609         #1
2610         \@@_adjust_size_box:
2611         \box_use_drop:N \l_@@_cell_box
2612     }
2613 }
2614 \int_gincr:N \c@jCol
2615 \@@_rec_preamble_after_col:n
2616 }

```

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

```

2617 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2618 {
2619     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2620     \tl_gclear:N \g_@@_pre_cell_tl
2621     \tl_gput_right:Nn \g_@@_array_preamble_tl
2622     {
2623         > {

```

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`.

```

2624         \setlength { \l_@@_col_width_dim } { #4 }
2625         \hbox_set:Nw \l_@@_cell_box
2626         \@@_cell_begin:
2627         \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2628     }
2629     c
2630     < {
2631         \@@_cell_end:
2632         \hbox_set_end:
2633         #1
2634         \@@_adjust_size_box:
2635         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2636     }
2637 }

```

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

```

2638     \int_gincr:N \c@jCol
2639     \@@_rec_preamble_after_col:n
2640 }

```

```

2641 \cs_new_protected:Npn \@@_special_W:
2642 {
2643     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \l_@@_col_width_dim }
2644     { \@@_warning:n { W-warning } }
2645 }

```

For `S` (of `siunitx`).

```

2646 \cs_new_protected:Npn \@@_S: #1 #2
2647 {

```

```

2648 \str_if_eq:nnTF { #2 } { [ ] }
2649 { \@@_make_preamble_S:w [ ] }
2650 { \@@_make_preamble_S:w [ ] { #2 } }
2651 }

2652 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2653 { \@@_make_preamble_S:i:n { #1 } }

2654 \cs_new_protected:Npn \@@_make_preamble_S:i:n #1
2655 {
2656 \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2657 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2658 \tl_gclear:N \g_@@_pre_cell_tl
2659 \tl_gput_right:Nn \g_@@_array_preamble_tl
2660 {
2661 > {

```

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).

```

2662 \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2663 \keys_set:nn { siunitx } { #1 }
2664 \@@_cell_begin:
2665 \siunitx_cell_begin:w
2666 }
2667 c
2668 <
2669 {
2670 \siunitx_cell_end:

```

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

```

2671 \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2672 {
2673 \bool_if:NTF \l__siunitx_table_text_bool
2674 { \bool_set_true:N }
2675 { \bool_set_false:N }
2676 \l__siunitx_table_text_bool
2677 }
2678 \@@_cell_end:
2679 }
2680 }

```

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

```

2681 \int_gincr:N \c@jCol
2682 \@@_rec_preamble_after_col:n
2683 }

```

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

```

2684 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2685 {
2686 \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.

```

2687 \int_if_zero:nTF { \c@jCol }
2688 {
2689 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2690 {

```

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

```

2691 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2692 \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl

```



```

2693         \@@_rec_preamble:n #2
2694     }
2695     {
2696         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2697         \@@_make_preamble_iv:nn { #1 } { #2 }
2698     }
2699 }
2700 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2701 }
2702 \cs_set_eq:cc { @@ _ \token_to_str:N [ : ] { @@ _ \token_to_str:N ( : }
2703 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2704 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2705 {
2706     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2707     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2708     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2709     {
2710         \@@_error:nn { delimiter~after~opening } { #2 }
2711         \@@_rec_preamble:n
2712     }
2713     { \@@_rec_preamble:n #2 }
2714 }

```

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

```

2715 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2716 { \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}`).

```

2717 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2718 {
2719     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2720     \tl_if_in:nnTF { ) ] \} } { #2 }
2721     { \@@_make_preamble_v:nnn #1 #2 }
2722     {
2723         \str_if_eq:nnTF { \s_stop } { #2 }
2724         {
2725             \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2726             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2727             {
2728                 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2729                 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2730                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2731                 \@@_rec_preamble:n #2
2732             }
2733         }
2734         {
2735             \tl_if_in:nnT { ( [ \{ \left } { #2 }
2736             { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2737             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2738             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2739             \@@_rec_preamble:n #2
2740         }
2741     }
2742 }
2743 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2744 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2745 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2746 {
2747     \str_if_eq:nnTF { \s_stop } { #3 }

```

```

2748 {
2749   \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2750   {
2751     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2752     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2753     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2754     \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2755   }
2756   {
2757     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2758     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2759     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2760     \@@_error:nn { double~closing~delimiter } { #2 }
2761   }
2762 }
2763 {
2764   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2765   { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2766   \@@_error:nn { double~closing~delimiter } { #2 }
2767   \@@_rec_preamble:n #3
2768 }
2769 }

2770 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2771 { \use:c { @@ _ \token_to_str:N ) : } }

```

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

```

2772 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2773 {
2774   \str_if_eq:nnTF { #1 } { < }
2775   { \@@_rec_preamble_after_col_i:n }
2776   {
2777     \str_if_eq:nnTF { #1 } { @ }
2778     { \@@_rec_preamble_after_col_ii:n }
2779     {
2780       \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2781       {
2782         \tl_gput_right:Nn \g_@@_array_preamble_tl
2783         { ! { \skip_horizontal:N \arrayrulewidth } }
2784       }
2785       {
2786         \clist_if_in:NeT \l_@@_vlines_clist
2787         { \int_eval:n { \c@jCol + 1 } }
2788         {
2789           \tl_gput_right:Nn \g_@@_array_preamble_tl
2790           { ! { \skip_horizontal:N \arrayrulewidth } }
2791         }
2792       }
2793       \@@_rec_preamble:n { #1 }
2794     }
2795   }
2796 }

2797 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2798 {
2799   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2800   \@@_rec_preamble_after_col:n
2801 }

```

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

```

2802 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2803 {
2804   \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2805   {
2806     \tl_gput_right:Nn \g_@@_array_preamble_tl
2807     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2808   }
2809   {
2810     \clist_if_in:NneTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2811     {
2812       \tl_gput_right:Nn \g_@@_array_preamble_tl
2813       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2814     }
2815     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2816   }
2817   \@@_rec_preamble:n
2818 }

2819 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2820 {
2821   \tl_clear:N \l_tmpa_tl
2822   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2823   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2824 }

```

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

```

2825 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2826 { \@@_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.

```

2827 \cs_new_protected:Npn \@@_X: #1 #2
2828 {
2829   \str_if_eq:nnTF { #2 } { [ ]
2830   { \@@_make_preamble_X:w [ ] }
2831   { \@@_make_preamble_X:w [ ] #2 }
2832 }

2833 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2834 { \@@_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`.

```

2835 \keys_define:nn { nicematrix / X-column }
2836 {
2837   V .code:n =
2838   \IfPackageLoadedTF { varwidth }
2839   {
2840     \bool_set_true:N \l_@@_V_of_X_bool
2841     \bool_gset_true:N \g_@@_V_of_X_bool
2842   }
2843   { \@@_error_or_warning:n { varwidth-not-loaded-in-X } } ,
2844   unknown .code:n =
2845   \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2846   { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2847   { \@@_error_or_warning:n { invalid-weight } }
2848 }

```

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

```
2849 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2850 {
```

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).

```
2851 \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).

```
2852 \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.

```
2853 \fp_set:Nn \l_tmpa_fp { 1.0 }
```

```
2854 \@@_keys_p_column:n { #1 }
```

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

```
2855 \bool_set_false:N \l_@@_V_of_X_bool
2856 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
```

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

```
2857 \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).

```
2858 \bool_if:NTF \l_@@_X_columns_aux_bool
2859 {
2860 \@@_make_preamble_ii_iv:nnn
```

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

```
2861 { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2862 { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
2863 { \@@_no_update_width: }
2864 }
```

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.

```
2865 {
2866 \tl_gput_right:Nn \g_@@_array_preamble_tl
2867 {
2868 > {
2869 \@@_cell_begin:
2870 \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.

```
2871 \NotEmpty
```

The following code will nullify the box of the cell.

```
2872 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2873 { \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.

```
2874 \begin { minipage } { 5 cm } \arraybackslash
2875 }
2876 c
2877 < {
2878 \end { minipage }
2879 \@@_cell_end:
```

```

2880         }
2881     }
2882     \int_gincr:N \c@jCol
2883     \@@_rec_preamble_after_col:n
2884 }
2885 }

2886 \cs_new_protected:Npn \@@_no_update_width:
2887 {
2888     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2889     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2890 }

```

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

```

2891 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2892 {
2893     \seq_gput_right:Ne \g_@@_cols_vlism_seq
2894     { \int_eval:n { \c@jCol + 1 } }
2895     \tl_gput_right:Ne \g_@@_array_preamble_tl
2896     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2897     \@@_rec_preamble:n
2898 }

```

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.

```

2899 \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).

```

2900 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2901 { \@@_fatal:n { Preamble-forgotten } }
2902 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
2903 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
2904 { @@ _ \token_to_str:N \hline : }
2905 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
2906 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
2907 { @@ _ \token_to_str:N \hline : }
2908 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
2909 { @@ _ \token_to_str:N \hline : }
2910 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
2911 { @@ _ \token_to_str:N \hline : }
2912 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
2913 { @@ _ \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).

```

2914 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2915 {

```

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

```

2916     \multispan { #1 }
2917     \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2918     \begingroup
2919     \tbl_update_multicolumn_cell_data:n { #1 }

```

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

```
2920 \tl_gclear:N \g_@@_preamble_tl
2921 \@@_make_m_preamble:n #2 \q_stop
```

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

```
2922 \def \@addamp
2923 {
2924   \legacy_if:nTF { @firstamp }
2925   { \legacy_if_set_false:n { @firstamp } }
2926   { \@preamerr 5 }
2927 }
2928 \exp_args:No \@mkpream \g_@@_preamble_tl
2929 \@addtopreamble \@empty
2930 \endgroup
2931 \UseTaggingSocket { tbl / colspan } { #1 }
```

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

```
2932 \int_compare:nNtT { #1 } > { \c_one_int }
2933 {
2934   \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2935   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2936   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2937   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2938   {
2939     {
2940       \int_if_zero:nTF { \c@jCol }
2941       { \int_eval:n { \c@iRow + 1 } }
2942       { \int_use:N \c@iRow }
2943     }
2944     { \int_eval:n { \c@jCol + 1 } }
2945     {
2946       \int_if_zero:nTF { \c@jCol }
2947       { \int_eval:n { \c@iRow + 1 } }
2948       { \int_use:N \c@iRow }
2949     }
2950     { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block.

```
2951 { }
2952 }
2953 }
```

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

```
2954 \RenewDocumentCommand { \cellcolor } { 0 { } m }
2955 {
2956   \tl_gput_right:Ne \g_@@_pre_code_before_tl
2957   {
2958     \@_rectanglecolor [ ##1 ]
2959     { \exp_not:n { ##2 } }
2960     { \int_use:N \c@iRow - \int_use:N \c@jCol }
2961     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2962   }
2963   \ignorespaces
2964 }
```

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

```
2965 \def \@sharp { #3 }
2966 \@arstrut
2967 \@preamble
2968 \null
```

We add some lines.

```

2969 \int_gadd:Nn \c@jCol { #1 - 1 }
2970 \int_compare:nNtT { \c@jCol } > { \g_@@_col_total_int }
2971 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2972 \ignorespaces
2973 }

```

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`.

```

2974 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2975 {
2976   \str_case:nnF { #1 }
2977   {
2978     c { \@@_make_m_preamble_i:n #1 }
2979     l { \@@_make_m_preamble_i:n #1 }
2980     r { \@@_make_m_preamble_i:n #1 }
2981     > { \@@_make_m_preamble_ii:nn #1 }
2982     ! { \@@_make_m_preamble_ii:nn #1 }
2983     @ { \@@_make_m_preamble_ii:nn #1 }
2984     | { \@@_make_m_preamble_iii:n #1 }
2985     p { \@@_make_m_preamble_iv:nnn t #1 }
2986     m { \@@_make_m_preamble_iv:nnn c #1 }
2987     b { \@@_make_m_preamble_iv:nnn b #1 }
2988     w { \@@_make_m_preamble_v:nnnn { } #1 }
2989     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2990     \q_stop { }
2991   }
2992   {
2993     \cs_if_exist:cTF { NC @ find @ #1 }
2994     {
2995       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2996       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2997     }
2998     {
2999       \str_if_eq:nnTF { #1 } { S }
3000       { \@@_fatal:n { unknown~column~type~S~multicolumn } }
3001       { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
3002     }
3003   }
3004 }

```

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

```

3005 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
3006 {
3007   \tl_gput_right:Nn \g_@@_preamble_tl
3008   {
3009     > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
3010     #1
3011     < \@@_cell_end:
3012   }

```

We test for the presence of a `<`.

```

3013 \@@_make_m_preamble_x:n
3014 }

```

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

```

3015 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
3016 {
3017   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
3018   \@@_make_m_preamble:n
3019 }

```

For l

```

3020 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
3021 {
3022   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
3023   \@@_make_m_preamble:n
3024 }

```

For p, m and b

```

3025 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
3026 {
3027   \tl_gput_right:Nn \g_@@_preamble_tl
3028   {
3029     > {
3030       \@@_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`.

```

3031       \setlength { \l_tmpa_dim } { #3 }
3032       \begin { minipage } [ #1 ] { \l_tmpa_dim }
3033       \mode_leave_vertical:
3034       \arraybackslash
3035       \vrule height \box_ht:N \@@arstrutbox depth \c_zero_dim width \c_zero_dim
3036     }
3037     c
3038     < {
3039       \vrule height \c_zero_dim depth \box_dp:N \@@arstrutbox width \c_zero_dim
3040       \end { minipage }
3041       \@@_cell_end:
3042     }
3043   }

```

We test for the presence of a `<`.

```

3044   \@@_make_m_preamble_x:n
3045 }

```

For w and W

```

3046 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3047 {
3048   \tl_gput_right:Nn \g_@@_preamble_tl
3049   {
3050     > {
3051       \dim_set:Nn \l_@@_col_width_dim { #4 }
3052       \hbox_set:Nw \l_@@_cell_box
3053       \@@_cell_begin:
3054       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3055     }
3056     c
3057     < {
3058       \@@_cell_end:
3059       \hbox_set_end:
3060       \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3061       #1
3062       \@@_adjust_size_box:
3063       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3064     }
3065   }

```

We test for the presence of a `<`.

```

3066   \@@_make_m_preamble_x:n
3067 }

```


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

```

3068 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3069 {
3070   \str_if_eq:nnTF { #1 } { < }
3071     { \@@_make_m_preamble_ix:n }
3072     { \@@_make_m_preamble:n { #1 } }
3073 }
3074 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3075 {
3076   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3077   \@@_make_m_preamble_x:n
3078 }

```

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`).

```

3079 \cs_new_protected:Npn \@@_put_box_in_flow:
3080 {
3081   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3082   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3083   \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
3084     { \box_use_drop:N \l_tmpa_box }
3085     { \@@_put_box_in_flow_i: }
3086 }

```

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

```

3087 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3088 {
3089   \pgfpicture
3090     \@@_qpoint:n { row - 1 }
3091     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3092     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3093     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3094     \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).

```

3095   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3096   {
3097     \int_set:Nn \l_tmpa_int
3098       { \str_range:Nnn \l_@@_baseline_tl { 6 } { -1 } }
3099     \bool_lazy_or:nnT
3100       { \int_compare_p:nNn { \l_tmpa_int } < { 1 } }
3101       { \int_compare_p:nNn { \l_tmpa_int } > { \c@iRow + 1 } }
3102     {
3103       \@@_error:n { bad-value-for-baseline-line }
3104       \int_set_eq:NN \l_tmpa_int \c_one_int
3105     }
3106     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3107   }
3108   {
3109     \str_if_eq:eeTF { \l_@@_baseline_tl } { t }
3110     { \int_set_eq:NN \l_tmpa_int \c_one_int }
3111     {
3112       \str_if_eq:onTF \l_@@_baseline_tl { b }
3113       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3114       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3115     }
3116     \bool_lazy_or:nnT

```

```

3117         { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3118         { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3119         {
3120             \@@_error:n { bad-value~for~baseline }
3121             \int_set_eq:NN \l_tmpa_int \c_one_int
3122         }
3123         \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3124         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3125     }
3126     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3127     \endpgfpicture
3128     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3129     \box_use_drop:N \l_tmpa_box
3130 }

```

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).

```

3131 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3132 {

```

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.

```

3133     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3134     {
3135         \int_compare:nNnT { \c@jCol } > { \c_one_int }
3136         {
3137             \box_set_wd:Nn \l_@@_the_array_box
3138             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3139         }
3140     }

```

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

```

3141     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3142     \bool_if:NT \l_@@_caption_above_bool
3143     {
3144         \tl_if_empty:NF \l_@@_caption_tl
3145         {
3146             \bool_set_false:N \g_@@_caption_finished_bool
3147             \int_gzero:N \c@tabularnote
3148             \@@_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).

```

3149         \int_compare:nNnT { \g_@@_notes_caption_int } > { \c_zero_int }
3150         {
3151             \tl_gput_right:Ne \g_@@_aux_tl
3152             {
3153                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3154                 { \int_use:N \g_@@_notes_caption_int }
3155             }
3156             \int_gzero:N \g_@@_notes_caption_int
3157         }
3158     }
3159 }

```

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

```

3160   \hbox
3161   {
3162   \box_use_drop:N \l_@@_the_array_box

```

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

```

3163   \@@_create_extra_nodes:
3164   \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3165 }

```

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).

```

3166   \bool_lazy_any:nT
3167   {
3168     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3169     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3170     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3171   }
3172   \@@_insert_tabularnotes:
3173   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3174   \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3175   \end { minipage }
3176 }

```

```

3177 \cs_new_protected:Npn \@@_insert_caption:
3178 {
3179   \tl_if_empty:NF \l_@@_caption_tl
3180   {
3181     \cs_if_exist:NTF \@capttype
3182     { \@@_insert_caption_i: }
3183     { \@@_error:n { caption~outside~float } }
3184   }
3185 }

```

```

3186 \cs_new_protected:Npn \@@_insert_caption_i:
3187 {
3188   \group_begin:

```

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

```

3189   \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.

```

3190   \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
3191   \tl_if_empty:NTF \l_@@_short_caption_tl
3192   { \caption }
3193   { \caption [ \l_@@_short_caption_tl ] }
3194   { \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.

```

3195   \bool_if:NF \g_@@_caption_finished_bool

```

```

3196     {
3197         \bool_gset_true:N \g_@@_caption_finished_bool
3198         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3199         \int_gzero:N \c@tabularnote
3200     }
3201     \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3202     \group_end:
3203 }
3204 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3205 {
3206     \@@_error_or_warning:n { tabularnote~below~the~tabular }
3207     \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3208 }
3209 \cs_new_protected:Npn \@@_insert_tabularnotes:
3210 {
3211     \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3212     \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3213     \skip_vertical:N 0.65ex

```

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

```

3214     \group_begin:
3215     \l_@@_notes_code_before_tl
3216     \tl_if_empty:NF \g_@@_tabularnote_tl
3217     {
3218         \g_@@_tabularnote_tl \par
3219         \tl_gclear:N \g_@@_tabularnote_tl
3220     }

```

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.

```

3221     \int_compare:nNnT { \c@tabularnote } > { \c_zero_int }
3222     {
3223         \bool_if:NTF \l_@@_notes_para_bool
3224         {
3225             \begin { tabularnotes* }
3226             \seq_map_inline:Nn \g_@@_notes_seq
3227             { \@@_one_tabularnote:nn ##1 }
3228             \strut
3229             \end { tabularnotes* }

```

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

```

3230         \par
3231     }
3232     {
3233         \tabularnotes
3234         \seq_map_inline:Nn \g_@@_notes_seq
3235         { \@@_one_tabularnote:nn ##1 }
3236         \strut
3237         \endtabularnotes
3238     }
3239 }
3240 \unskip
3241 \group_end:
3242 \bool_if:NT \l_@@_notes_bottomrule_bool
3243 {
3244     \IfPackageLoadedTF { booktabs }
3245     {

```

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

```

3246         \skip_vertical:N \aboverulesep

```

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

```

3247         { \CT@arc@ \hrule height \heavyrulewidth }

```

```

3248     }
3249     { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3250   }
3251   \l_@@_notes_code_after_tl
3252   \seq_gclear:N \g_@@_notes_seq
3253   \seq_gclear:N \g_@@_notes_in_caption_seq
3254   \int_gzero:N \c@tabularnote
3255 }

```

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.

```

3256 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3257 {
3258   \tl_if_novalue:nTF { #1 }
3259   { \item }
3260   { \item [ \@@_notes_label_in_list:n { #1 } ] }
3261 }

```

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.

```

3262 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3263 {
3264   \pgfpicture
3265     \@@_qpoint:n { row - 1 }
3266     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3267     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3268     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3269   \endpgfpicture
3270   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3271   \int_if_zero:nT { \l_@@_first_row_int }
3272   {
3273     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3274     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3275   }
3276   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3277 }

```

Now, the general case.

```

3278 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3279 {

```

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

```

3280   \str_if_eq:eeT { \l_@@_baseline_tl } { t }
3281   { \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`.

```

3282   \pgfpicture
3283     \@@_qpoint:n { row - 1 }
3284     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3285     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3286     {
3287       \int_set:Nn \l_tmpa_int
3288       {
3289         \str_range:Nnn
3290         \l_@@_baseline_tl
3291         { 6 }
3292         { \tl_count:o \l_@@_baseline_tl }
3293       }
3294     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }

```

```

3295 }
3296 {
3297   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3298   \bool_lazy_or:nnT
3299     { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3300     { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3301     {
3302       \@@_error:n { bad-value-for-baseline }
3303       \int_set:Nn \l_tmpa_int 1
3304     }
3305   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3306 }
3307 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3308 \endpgfpicture
3309 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3310 \int_if_zero:nT { \l_@@_first_row_int }
3311 {
3312   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3313   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3314 }
3315 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3316 }

```

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.

```

3317 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3318 {

```

We will compute the real width of both delimiters used.

```

3319   \dim_zero_new:N \l_@@_real_left_delim_dim
3320   \dim_zero_new:N \l_@@_real_right_delim_dim
3321   \hbox_set:Nn \l_tmpb_box
3322   {
3323     \m@th
3324     $ % $
3325     \left #1
3326     \vcenter
3327     {
3328       \vbox_to_ht:nn
3329         { \box_ht_plus_dp:N \l_tmpa_box }
3330         { }
3331     }
3332     \right .
3333     $ % $
3334   }
3335   \dim_set:Nn \l_@@_real_left_delim_dim
3336     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3337   \hbox_set:Nn \l_tmpb_box
3338   {
3339     \m@th
3340     $ % $
3341     \left .
3342     \vbox_to_ht:nn
3343       { \box_ht_plus_dp:N \l_tmpa_box }
3344       { }
3345     \right #2
3346     $ % $
3347   }
3348   \dim_set:Nn \l_@@_real_right_delim_dim
3349     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3350   \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }

```

```

3351 \@@_put_box_in_flow:
3352 \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3353 }

```

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).

```

3354 \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).

```

3355 {
3356   \peek_remove_spaces:n
3357   {
3358     \peek_meaning:NTF \end
3359     { \@@_analyze_end:Nn }
3360     {
3361       \@@_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`).

```

3362       \@@_array:o \g_@@_array_preamble_tl
3363     }
3364   }
3365 }
3366 {
3367   \@@_create_col_nodes:
3368   \endarray
3369 }

```

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

```

3370 \NewDocumentEnvironment { @@-light-syntax } { b }
3371 {

```

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`.

```

3372 \tl_if_empty:nT { #1 }
3373 { \@@_fatal:n { empty-environment } }
3374 \tl_if_in:nnT { #1 } { & }
3375 { \@@_fatal:n { ampersand-in~light-syntax } }
3376 \tl_if_in:nnT { #1 } { \ }
3377 { \@@_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`.

```

3378 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3379 }

```

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.

```

3380 {

```

```

3381 \@@_create_col_nodes:
3382 \endarray
3383 }
3384 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3385 {
3386 \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).

```

3387 \seq_clear_new:N \l_@@_rows_seq

```

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

```

3388 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3389 \bool_if:NTF \l_@@_light_syntax_expanded_bool
3390 { \seq_set_split:Nee }
3391 { \seq_set_split:Non }
3392 \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3393 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3394 \tl_if_empty:NF \l_tmpa_tl
3395 { \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.

```

3396 \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
3397 { \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`).

```

3398 \tl_build_begin:N \l_@@_new_body_tl
3399 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3400 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3401 \@@_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).

```

3402 \seq_map_inline:Nn \l_@@_rows_seq
3403 {
3404 \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3405 \@@_line_with_light_syntax:n { ##1 }
3406 }
3407 \tl_build_end:N \l_@@_new_body_tl
3408 \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
3409 {
3410 \int_set:Nn \l_@@_last_col_int
3411 { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3412 }

```

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.

```

3413 \@@_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`).

```

3414 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3415 }

```



```

3416 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3417 {
3418   \seq_clear_new:N \l_@@_cells_seq
3419   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3420   \int_set:Nn \l_@@_nb_cols_int
3421   {
3422     \int_max:nn
3423     { \l_@@_nb_cols_int }
3424     { \seq_count:N \l_@@_cells_seq }
3425   }
3426   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3427   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3428   \seq_map_inline:Nn \l_@@_cells_seq
3429   { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3430 }
3431 \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`.

```

3432 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3433 {
3434   \str_if_eq:eeT { \g_@@_name_env_str } { #2 }
3435   { \@@_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.

```

3436   \end { #2 }
3437 }

```

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`).

```

3438 \cs_new:Npn \@@_create_col_nodes:
3439 {
3440   \crrr
3441   \int_if_zero:nT { \l_@@_first_col_int }
3442   {
3443     \omit
3444     \hbox_overlap_left:n
3445     {
3446       \bool_if:NT \l_@@_code_before_bool
3447       { \pgfsys@markposition { \@@_env: - col - 0 } }
3448       \pgfpicture
3449       \pgfrememberpicturepositiononpagetrue
3450       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3451       \str_if_empty:NF \l_@@_name_str
3452       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3453       \endpgfpicture
3454       \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3455     }
3456     &
3457   }
3458   \omit

```

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

```

3459   \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`).

```

3460   \int_if_zero:nTF { \l_@@_first_col_int }
3461   {
3462     \@@_mark_position:n { 1 }

```

```

3463 \pgfpicture
3464 \pgfrememberpicturepositiononpagetrue
3465 \pgfcoordinate { \@@_env: - col - 1 }
3466 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3467 \str_if_empty:NF \l_@@_name_str
3468 { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3469 \endpgfpicture
3470 }
3471 {
3472 \bool_if:NT \l_@@_code_before_bool
3473 {
3474 \hbox
3475 {
3476 \skip_horizontal:n { 0.5 \arrayrulewidth }
3477 \pgfsys@markposition { \@@_env: - col - 1 }
3478 \skip_horizontal:n { -0.5 \arrayrulewidth }
3479 }
3480 }
3481 \pgfpicture
3482 \pgfrememberpicturepositiononpagetrue
3483 \pgfcoordinate { \@@_env: - col - 1 }
3484 { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3485 \@@_node_alias:n { 1 }
3486 \endpgfpicture
3487 }

```

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.

```

3488 \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
3489 \bool_if:NF \l_@@_auto_columns_width_bool
3490 { \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim } }
3491 {
3492 \bool_lazy_and:nnTF
3493 { \l_@@_auto_columns_width_bool }
3494 { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3495 { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3496 { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3497 \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3498 }
3499 \skip_horizontal:N \g_tmpa_skip
3500 \hbox
3501 {
3502 \@@_mark_position:n { 2 }
3503 \pgfpicture
3504 \pgfrememberpicturepositiononpagetrue
3505 \pgfcoordinate { \@@_env: - col - 2 }
3506 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3507 \@@_node_alias:n { 2 }
3508 \endpgfpicture
3509 }

```

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.

```

3510 \int_gset_eq:NN \g_tmpa_int \c_one_int
3511 \bool_if:NTF \g_@@_last_col_found_bool
3512 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3513 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3514 {
3515 &
3516 \omit

```

```
3517 \int_gincr:N \g_tmpa_int
```

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

```
3518 \skip_horizontal:N \g_tmpa_skip
3519 \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }
```

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

```
3520 \pgfpicture
3521 \pgfrememberpicturepositiononpagetrue
3522 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3523 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3524 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3525 \endpgfpicture
3526 }
```

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).

```
3527 \bool_lazy_or:nnF
3528 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3529 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3530 {
3531   &
3532   \omit
3533   \skip_horizontal:N \g_tmpa_skip
3534   \int_gincr:N \g_tmpa_int
3535   \bool_lazy_any:nF
3536   {
3537     \g_@@_delims_bool
3538     \l_@@_tabular_bool
3539     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3540     \l_@@_exterior_arraycolsep_bool
3541     \l_@@_bar_at_end_of_pream_bool
3542   }
3543   { \skip_horizontal:n { - \col@sep } }
3544   \bool_if:NT \l_@@_code_before_bool
3545   {
3546     \hbox
3547     {
3548       \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.

```
3549 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3550 { \skip_horizontal:n { - \arraycolsep } }
3551 \pgfsys@markposition
3552 { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3553 \skip_horizontal:n { 0.5 \arrayrulewidth }
3554 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3555 { \skip_horizontal:N \arraycolsep }
3556 }
3557 }
3558 \pgfpicture
3559 \pgfrememberpicturepositiononpagetrue
3560 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3561 {
3562   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3563   {
3564     \pgfpoint
3565     { - 0.5 \arrayrulewidth - \arraycolsep }
3566     \c_zero_dim
3567   }
3568   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3569 }
3570 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
```

```

3571         \endpgfpicture
3572     }

3573 \bool_if:NT \g_@@_last_col_found_bool
3574 {
3575     \hbox_overlap_right:n
3576     {
3577         \skip_horizontal:N \g_@@_width_last_col_dim
3578         \skip_horizontal:N \col@sep
3579         \bool_if:NT \l_@@_code_before_bool
3580         {
3581             \pgfsys@markposition
3582             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3583         }
3584         \pgfpicture
3585         \pgfrememberpicturepositiononpagetrue
3586         \pgfcoordinate
3587         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3588         \pgfpointorigin
3589         \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3590         \endpgfpicture
3591     }
3592 }
3593 }

3594 \cs_new_protected:Npn \@@_mark_position:n #1
3595 {
3596     \bool_if:NT \l_@@_code_before_bool
3597     {
3598         \hbox
3599         {
3600             \skip_horizontal:n { -0.5 \arrayrulewidth }
3601             \pgfsys@markposition { \@@_env: - col - #1 }
3602             \skip_horizontal:n { 0.5 \arrayrulewidth }
3603         }
3604     }
3605 }

3606 \cs_new_protected:Npn \@@_node_alias:n #1
3607 {
3608     \str_if_empty:NF \l_@@_name_str
3609     { \pgfnodealias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3610 }

```

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

```

3611 \tl_const:Nn \c_@@_preamble_first_col_tl
3612 {
3613     >
3614     {

```

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

```

3615         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3616         \bool_gset_true:N \g_@@_after_col_zero_bool
3617         \@@_begin_of_row:
3618         \hbox_set:Nw \l_@@_cell_box
3619         \@@_math_toggle:
3620         \@@_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”.

```

3621     \int_compare:nNt { \c@iRow } > { \c_zero_int }
3622     {
3623       \bool_lazy_or:nnT
3624       { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3625       { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3626       {
3627         \l_@@_code_for_first_col_tl
3628         \xglobal \colorlet { nicematrix-first-col } { . }
3629       }
3630     }
3631 }

```

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`.

```

3632   1
3633   <
3634   {
3635     \@@_math_toggle:
3636     \hbox_set_end:
3637     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3638     \@@_adjust_size_box:
3639     \@@_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.

```

3640     \dim_gset:Nn \g_@@_width_first_col_dim
3641     { \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.

```

3642     \hbox_overlap_left:n
3643     {
3644       \dim_compare:nNtF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3645       { \@@_node_cell: }
3646       { \box_use_drop:N \l_@@_cell_box }
3647       \skip_horizontal:N \l_@@_left_delim_dim
3648       \skip_horizontal:N \l_@@_left_margin_dim
3649       \skip_horizontal:N \l_@@_extra_left_margin_dim
3650     }
3651     \bool_gset_false:N \g_@@_empty_cell_bool
3652     \skip_horizontal:n { -2 \col@sep }
3653   }
3654 }

```

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

```

3655 \tl_const:Nn \c_@@_preamble_last_col_tl
3656 {
3657   >
3658   {
3659     \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).

```

3660     \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.

```

3661     \bool_gset_true:N \g_@@_last_col_found_bool
3662     \int_gincr:N \c@jCol
3663     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3664     \hbox_set:Nw \l_@@_cell_box
3665     \@@_math_toggle:
3666     \@@_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”.

```

3667     \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3668     {
3669         \bool_lazy_or:nnT
3670         { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3671         { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3672         {
3673             \l_@@_code_for_last_col_tl
3674             \xglobal \colorlet { nicematrix-last-col } { . }
3675         }
3676     }
3677 }
3678 1
3679 <
3680 {
3681     \@@_math_toggle:
3682     \hbox_set_end:
3683     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3684     \@@_adjust_size_box:
3685     \@@_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.

```

3686     \dim_gset:Nn \g_@@_width_last_col_dim
3687     { \dim_max:nn { \g_@@_width_last_col_dim } { \box_wd:N \l_@@_cell_box } }
3688     \skip_horizontal:n { -2 \col@sep }

```

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

```

3689     \hbox_overlap_right:n
3690     {
3691         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3692         {
3693             \skip_horizontal:N \l_@@_right_delim_dim
3694             \skip_horizontal:N \l_@@_right_margin_dim
3695             \skip_horizontal:N \l_@@_extra_right_margin_dim
3696             \@@_node_cell:
3697         }
3698     }
3699     \bool_gset_false:N \g_@@_empty_cell_bool
3700 }
3701 }

```

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

```

3702 \NewDocumentEnvironment { NiceArray } { }
3703 {
3704     \bool_gset_false:N \g_@@_delims_bool
3705     \str_if_empty:NT \g_@@_name_env_str
3706     { \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).

```

3707     \NiceArrayWithDelims . .
3708 }
3709 { \endNiceArrayWithDelims }

```

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

```

3710 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3711 {
3712     \NewDocumentEnvironment { #1 NiceArray } { }
3713     {

```

```

3714 \bool_gset_true:N \g_@@_delims_bool
3715 \str_if_empty:NT \g_@@_name_env_str
3716 { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3717 \@@_test_if_math_mode:
3718 \NiceArrayWithDelims #2 #3
3719 }
3720 { \endNiceArrayWithDelims }
3721 }
3722 \@@_def_env:NNN p ( )
3723 \@@_def_env:NNN b [ ]
3724 \@@_def_env:NNN B \{ \}
3725 \@@_def_env:NNN v \vert \vert
3726 \@@_def_env:NNN V \Vert \Vert

```

13 The environment `{NiceMatrix}` and its variants

```

3727 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3728 {
3729 \bool_set_false:N \l_@@_preamble_bool
3730 \tl_clear:N \l_tmpa_tl
3731 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3732 { \tl_set:Nn \l_tmpa_tl { @ { } } }
3733 \tl_put_right:Nn \l_tmpa_tl
3734 {
3735 *
3736 {
3737 \int_case:nnF \l_@@_last_col_int
3738 {
3739 { -2 } { \c@MaxMatrixCols }
3740 { -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).

```

3741 }
3742 { \int_eval:n { \l_@@_last_col_int - 1 } }
3743 }
3744 { #2 }
3745 }
3746 \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3747 \exp_args:No \l_tmpb_tl \l_tmpa_tl
3748 }
3749 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3750 \clist_map_inline:nn { p , b , B , v , V }
3751 {
3752 \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3753 {
3754 \bool_gset_true:N \g_@@_delims_bool
3755 \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3756 \int_if_zero:nT { \l_@@_last_col_int }
3757 {
3758 \bool_set_true:N \l_@@_last_col_without_value_bool
3759 \int_set:Nn \l_@@_last_col_int { -1 }
3760 }
3761 \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3762 \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3763 }
3764 { \use:c { end #1 NiceArray } }
3765 }

```

We define also an environment `{NiceMatrix}`

```

3766 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3767 {
3768   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3769   \int_if_zero:nT { \l_@@_last_col_int }
3770   {
3771     \bool_set_true:N \l_@@_last_col_without_value_bool
3772     \int_set:Nn \l_@@_last_col_int { -1 }
3773   }
3774   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3775   \bool_lazy_or:nnT
3776     { \clist_if_empty_p:N \l_@@_vlines_clist }
3777     { \l_@@_except_borders_bool }
3778     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3779   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3780 }
3781 { \endNiceArray }

```

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

```

3782 \cs_new_protected:Npn \@@_NotEmpty:
3783 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3784 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3785 {

```

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.

```

3786   \dim_compare:nNnT { \l_@@_width_dim } = { \c_zero_dim }
3787   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3788   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3789   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3790   \tl_if_empty:NF \l_@@_short_caption_tl
3791   {
3792     \tl_if_empty:NT \l_@@_caption_tl
3793     {
3794       \@@_error_or_warning:n { short-caption-without-caption }
3795       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3796     }
3797   }
3798   \tl_if_empty:NF \l_@@_label_tl
3799   {
3800     \tl_if_empty:NT \l_@@_caption_tl
3801     { \@@_error_or_warning:n { label-without-caption } }
3802   }
3803   \NewDocumentEnvironment { TabularNote } { b }
3804   {
3805     \bool_if:NTF \l_@@_in_code_after_bool
3806     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3807     {
3808       \tl_if_empty:NF \g_@@_tabularnote_tl
3809       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3810       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3811     }
3812   }
3813   { }
3814   \@@_settings_for_tabular:
3815   \NiceArray { #2 }
3816 }
3817 { \endNiceArray }
3818 \cs_new_protected:Npn \@@_settings_for_tabular:
3819 {

```



```

3820 \bool_set_true:N \l_@@_tabular_bool
3821 \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3822 \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3823 \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3824 }

3825 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3826 {
3827 \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3828 \dim_set:Nn \l_@@_width_dim { #1 }
3829 \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3830 \@@_settings_for_tabular:
3831 \NiceArray { #3 }
3832 }
3833 {
3834 \endNiceArray
3835 \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3836 { \@@_error:n { NiceTabularX~without~X } }
3837 }

3838 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3839 {
3840 \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3841 \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3842 \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3843 \@@_settings_for_tabular:
3844 \NiceArray { #3 }
3845 }
3846 { \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`).

```

3847 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3848 {
3849 \bool_lazy_all:nT
3850 {
3851 { \dim_compare_p:nNn { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim } }
3852 { \l_@@_hvlines_bool }
3853 { ! \g_@@_delims_bool }
3854 { ! \l_@@_except_borders_bool }
3855 }
3856 {
3857 \bool_set_true:N \l_@@_except_borders_bool
3858 \clist_if_empty:NF \l_@@_corners_clist
3859 { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3860 \tl_gput_right:Nn \g_@@_pre_code_after_tl
3861 {
3862 \@@_stroke_block:nnn
3863 {
3864 rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3865 draw = \l_@@_rules_color_tl
3866 }
3867 { 1-1 }
3868 { \int_use:N \c@iRow - \int_use:N \c@jCol }
3869 }
3870 }
3871 }

```

```

3872 \cs_new_protected:Npn \@@_after_array:
3873 {

```

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

```

3874 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3875 \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.

```

3876 \bool_if:NT \g_@@_last_col_found_bool
3877 { \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`.

```

3878 \bool_if:NT \l_@@_last_col_without_value_bool
3879 { \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.

```

3880 \bool_if:NT \l_@@_last_row_without_value_bool
3881 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3882 \tl_gput_right:Ne \g_@@_aux_tl
3883 {
3884   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3885   {
3886     \int_use:N \l_@@_first_row_int ,
3887     \int_use:N \c@iRow ,
3888     \int_use:N \g_@@_row_total_int ,
3889     \int_use:N \l_@@_first_col_int ,
3890     \int_use:N \c@jCol ,
3891     \int_use:N \g_@@_col_total_int
3892   }
3893 }

```

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`).

```

3894 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3895 {
3896   \tl_gput_right:Ne \g_@@_aux_tl
3897   {
3898     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3899     { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }
3900   }
3901 }
3902 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3903 {
3904   \tl_gput_right:Ne \g_@@_aux_tl
3905   {
3906     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3907     { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
3908     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3909     { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
3910   }
3911 }

```

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

```
3912 \@@_create_diag_nodes:
```

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

```
3913 \pgfpicture
3914 \@@_create_aliases_last:
3915 \str_if_empty:NF \l_@@_name_str { \@@_create_alias_nodes: }
3916 \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.

```
3917 \bool_if:NT \l_@@_parallelize_diags_bool
3918 {
3919   \int_gzero:N \g_@@_ddots_int
3920   \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.

```
3921   \dim_gzero:N \g_@@_delta_x_one_dim
3922   \dim_gzero:N \g_@@_delta_y_one_dim
3923   \dim_gzero:N \g_@@_delta_x_two_dim
3924   \dim_gzero:N \g_@@_delta_y_two_dim
3925 }
3926 \bool_set_false:N \l_@@_initial_open_bool
3927 \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.

```
3928 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }
```

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

```
3929 \@@_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.

```
3930 \clist_if_empty:NF \l_@@_corners_clist
3931 {
3932   \bool_if:NTF \l_@@_no_cell_nodes_bool
3933     { \@@_error:n { corners~with~no~cell~nodes } }
3934     { \@@_compute_corners: }
3935 }
```

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

```
3936 \seq_gconcat:NNN \g_@@_pos_of_blocks_seq
3937   \g_@@_pos_of_blocks_seq
3938   \g_@@_future_pos_of_blocks_seq
3939 \seq_gclear:N \g_@@_future_pos_of_blocks_seq
```

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

```
3940 \@@_adjust_pos_of_blocks_seq:
```

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

```

3941 \@@_deal_with_rounded_corners:
3942 \clist_if_empty:NF \l_@@_hlines_clist { \@@_draw_hlines: }
3943 \clist_if_empty:NF \l_@@_vlines_clist { \@@_draw_vlines: }

```

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

```

3944 \IfPackageLoadedT { tikz }
3945 {
3946   \tikzset
3947   {
3948     every-picture / .style =
3949     {
3950       overlay ,
3951       remember~picture ,
3952       name~prefix = \@@_env: -
3953     }
3954   }
3955 }
3956 \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
3957 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3958 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3959 \cs_set_eq:NN \OverBrace \@@_OverBrace
3960 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3961 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3962 \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.

```

3963 \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
3964 \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.

```

3965 \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).

```

3966 \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`).

```

3967 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3968 { \@@_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:`.

```

3969 \bool_set_true:N \l_@@_in_code_after_bool
3970 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3971 \scan_stop:
3972 \tl_gclear:N \g_nicematrix_code_after_tl
3973 \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.

```

3974 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3975 \tl_if_empty:NF \g_@@_pre_code_before_tl
3976 {
3977   \tl_gput_right:Ne \g_@@_aux_tl
3978   {
3979     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl

```

```

3980         { \exp_not:o \g_@@_pre_code_before_tl }
3981     }
3982     \tl_gclear:N \g_@@_pre_code_before_tl
3983 }
3984 \tl_if_empty:NF \g_nicematrix_code_before_tl
3985 {
3986     \tl_gput_right:Ne \g_@@_aux_tl
3987     {
3988         \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3989         { \exp_not:o \g_nicematrix_code_before_tl }
3990     }
3991     \tl_gclear:N \g_nicematrix_code_before_tl
3992 }

3993 \str_gclear:N \g_@@_name_env_str
3994 \@@_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`.

```

3995     \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3996 }

3997 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
3998 {
3999     \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
4000     \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }

4001     \dim_set:Nn \l_@@_xdots_shorten_start_dim
4002     { 0.6 \l_@@_xdots_shorten_start_dim }
4003     \dim_set:Nn \l_@@_xdots_shorten_end_dim
4004     { 0.6 \l_@@_xdots_shorten_end_dim }
4005 }

```

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`.

```

4006 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4007 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

4008 \cs_new_protected:Npn \@@_create_alias_nodes:
4009 {
4010     \int_step_inline:nn { \c@iRow }
4011     {
4012         \pgfnodealias
4013         { \l_@@_name_str - ##1 - last }
4014         { \@@_env: - ##1 - \int_use:N \c@jCol }
4015     }
4016     \int_step_inline:nn { \c@jCol }
4017     {
4018         \pgfnodealias
4019         { \l_@@_name_str - last - ##1 }
4020         { \@@_env: - \int_use:N \c@iRow - ##1 }
4021     }
4022     \pgfnodealias

```

¹³e.g. `\color[rgb]{0.5,0.5,0}`

```

4023 { \l_@@_name_str - last - last }
4024 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
4025 }

```

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).

```

4026 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4027 {
4028   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4029   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4030 }

```

The following command must *not* be protected.

```

4031 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4032 {
4033   { #1 }
4034   { #2 }
4035   {
4036     \int_compare:nNnTF { #3 } > { 98 }
4037     { \int_use:N \c@iRow }
4038     { #3 }
4039   }
4040   {
4041     \int_compare:nNnTF { #4 } > { 98 }
4042     { \int_use:N \c@jCol }
4043     { #4 }
4044   }
4045   { #5 }
4046 }

```

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).

```

4047 \hook_gput_code:nnn { begindocument } { . }
4048 {
4049   \cs_new_protected:Npe \@@_draw_dotted_lines:
4050   {
4051     \c_@@_pgfortikzpicture_tl
4052     \@@_draw_dotted_lines_i:
4053     \c_@@_endpgfortikzpicture_tl
4054   }
4055 }

```

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

```

4056 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4057 {
4058   \pgfrememberpicturepositiononpagetrue
4059   \pgf@relevantforpicturesizefalse
4060   \g_@@_HVdotsfor_lines_tl
4061   \g_@@_Vdots_lines_tl
4062   \g_@@_Ddots_lines_tl
4063   \g_@@_Idots_lines_tl
4064   \g_@@_Cdots_lines_tl
4065   \g_@@_Ldots_lines_tl
4066 }

```

```

4067 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4068 {
4069   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4070   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4071 }

```

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

```

4072 \pgfdeclareshape { @@_diag_node }
4073 {
4074   \savedanchor { \five }
4075   {
4076     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4077     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4078   }
4079   \anchor { 5 } { \five }
4080   \anchor { center } { \pgfpointorigin }
4081   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4082   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4083   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4084   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4085   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4086   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4087   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4088   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4089   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4090   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4091 }

```

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).

```

4092 \cs_new_protected:Npn \@@_create_diag_nodes:
4093 {
4094   \pgfpicture
4095   \pgfrememberpicturepositiononpagetrue
4096   \int_step_inline:nn { \int_max:nn { \c@iRow } { \c@jCol } } {
4097     {
4098       \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4099       \dim_set_eq:NN \l_tmpa_dim \pgf@x
4100       \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4101       \dim_set_eq:NN \l_tmpb_dim \pgf@y
4102       \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4103       \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4104       \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4105       \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4106       \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.

```

4107       \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4108       \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4109       \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4110       \str_if_empty:NF \l_@@_name_str
4111       { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4112     }

```

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

```

4113       \int_set:Nn \l_tmpa_int { \int_max:nn { \c@iRow } { \c@jCol } + 1 }
4114       \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4115       \dim_set_eq:NN \l_tmpa_dim \pgf@y
4116       \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4117       \pgfcoordinate

```

```

4118     { \l_@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4119 \pgfnodealias
4120     { \l_@@_env: - last }
4121     { \l_@@_env: - \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
4122 \str_if_empty:NF \l_@@_name_str
4123     {
4124         \pgfnodealias
4125             { \l_@@_name_str - \int_use:N \l_tmpa_int }
4126             { \l_@@_env: - \int_use:N \l_tmpa_int }
4127         \pgfnodealias
4128             { \l_@@_name_str - last }
4129             { \l_@@_env: - last }
4130     }
4131 \endpgfpicture
4132 }

```

16 We draw the dotted lines

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

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

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

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

This command computes:

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

```

4133 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4134 {

```

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

```

4135 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4136 \int_set:Nn \l_@@_initial_i_int { #1 }
4137 \int_set:Nn \l_@@_initial_j_int { #2 }
4138 \int_set:Nn \l_@@_final_i_int { #1 }
4139 \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.

```

4140 \bool_set_false:N \l_@@_stop_loop_bool
4141 \bool_do_until:Nn \l_@@_stop_loop_bool
4142 {
4143   \int_add:Nn \l_@@_final_i_int { #3 }
4144   \int_add:Nn \l_@@_final_j_int { #4 }
4145   \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.

```

4146 \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4147 \if_int_compare:w #3 = \c_one_int
4148   \bool_set_true:N \l_@@_final_open_bool
4149 \else:
4150   \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4151   \bool_set_true:N \l_@@_final_open_bool
4152 \fi:
4153 \fi:
4154 \else:
4155   \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4156   \if_int_compare:w #4 = -1
4157     \bool_set_true:N \l_@@_final_open_bool
4158   \fi:
4159 \else:
4160   \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4161   \if_int_compare:w #4 = \c_one_int
4162     \bool_set_true:N \l_@@_final_open_bool
4163   \fi:
4164 \fi:
4165 \fi:
4166 \fi:
4167 \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.

```

4168 {

```

We do a step backwards.

```

4169   \int_sub:Nn \l_@@_final_i_int { #3 }
4170   \int_sub:Nn \l_@@_final_j_int { #4 }
4171   \bool_set_true:N \l_@@_stop_loop_bool
4172 }

```

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`.

```

4173 {
4174   \cs_if_exist:cTF
4175   {
4176     @@ _ dotted _
4177     \int_use:N \l_@@_final_i_int -
4178     \int_use:N \l_@@_final_j_int
4179   }
4180   {
4181     \int_sub:Nn \l_@@_final_i_int { #3 }
4182     \int_sub:Nn \l_@@_final_j_int { #4 }
4183     \bool_set_true:N \l_@@_final_open_bool
4184     \bool_set_true:N \l_@@_stop_loop_bool
4185   }
4186   {
4187     \cs_if_exist:cTF
4188     {
4189       pgf @ sh @ ns @ \@@_env:
4190       - \int_use:N \l_@@_final_i_int

```

```

4191         - \int_use:N \l_@@_final_j_int
4192     }
4193     { \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.

```

4194     {
4195         \cs_set_nopar:cpn
4196         {
4197             @@ _ dotted _
4198             \int_use:N \l_@@_final_i_int -
4199             \int_use:N \l_@@_final_j_int
4200         }
4201         { }
4202     }
4203 }
4204 }
4205 }

```

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

```

4206     \bool_set_false:N \l_@@_stop_loop_bool

```

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

```

4207     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4208     \bool_do_until:Nn \l_@@_stop_loop_bool
4209     {
4210         \int_sub:Nn \l_@@_initial_i_int { #3 }
4211         \int_sub:Nn \l_@@_initial_j_int { #4 }
4212         \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.

```

4213     \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4214     \if_int_compare:w #3 = \c_one_int
4215         \bool_set_true:N \l_@@_initial_open_bool
4216     \else:

```

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

```

4217         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4218         \bool_set_true:N \l_@@_initial_open_bool
4219     \fi:
4220     \fi:
4221     \else:
4222         \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4223         \if_int_compare:w #4 = \c_one_int
4224             \bool_set_true:N \l_@@_initial_open_bool
4225         \fi:
4226     \else:
4227         \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4228         \if_int_compare:w #4 = -1
4229             \bool_set_true:N \l_@@_initial_open_bool
4230         \fi:
4231     \fi:
4232     \fi:
4233     \fi:

```

```

4234 \bool_if:NTF \l_@@_initial_open_bool
4235 {
4236   \int_add:Nn \l_@@_initial_i_int { #3 }
4237   \int_add:Nn \l_@@_initial_j_int { #4 }
4238   \bool_set_true:N \l_@@_stop_loop_bool
4239 }
4240 {
4241   \cs_if_exist:cTF
4242   {
4243     @@ _ dotted _
4244     \int_use:N \l_@@_initial_i_int -
4245     \int_use:N \l_@@_initial_j_int
4246   }
4247   {
4248     \int_add:Nn \l_@@_initial_i_int { #3 }
4249     \int_add:Nn \l_@@_initial_j_int { #4 }
4250     \bool_set_true:N \l_@@_initial_open_bool
4251     \bool_set_true:N \l_@@_stop_loop_bool
4252   }
4253   {
4254     \cs_if_exist:cTF
4255     {
4256       pgf @ sh @ ns @ \@@_env:
4257       - \int_use:N \l_@@_initial_i_int
4258       - \int_use:N \l_@@_initial_j_int
4259     }
4260     { \bool_set_true:N \l_@@_stop_loop_bool }
4261     {
4262       \cs_set_nopar:cpn
4263       {
4264         @@ _ dotted _
4265         \int_use:N \l_@@_initial_i_int -
4266         \int_use:N \l_@@_initial_j_int
4267       }
4268       { }
4269     }
4270   }
4271 }
4272 }

```

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.

```

4273 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4274 {
4275   { \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`.

```

4276 { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4277 { \int_use:N \l_@@_final_i_int }
4278 { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4279 { }
4280 }
4281 }

```

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

```

4282 \cs_new_protected:Npn \@@_open_shorten:
4283 {

```

```

4284 \bool_if:NT \l_@@_initial_open_bool
4285 { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4286 \bool_if:NT \l_@@_final_open_bool
4287 { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4288 }

```

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).

```

4289 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4290 {
4291   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4292   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4293   \int_set_eq:NN \l_@@_row_max_int \c_iRow
4294   \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`.

```

4295 \seq_if_empty:NF \g_@@_submatrix_seq
4296 {
4297   \seq_map_inline:Nn \g_@@_submatrix_seq
4298   { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4299 }
4300 }

```

#1 and #2 are the numbers of row and columns of the cell where the command of dotted line (ex.: `\Vdots`) has been issued. #3, #4, #5 and #6 are the specification (in *i* and *j*) of the submatrix we are analyzing.

Here is the programmation of that command with the the standard syntax of L3.

```

\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
{
  \bool_if:nT
  {
    \int_compare_p:n { #3 <= #1 <= #5 }
    &&
    \int_compare_p:n { #4 <= #2 <= #6 }
  }
  {
    \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
    \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
    \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
    \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
  }
}

```

However, for efficiency, we will use the following version.

```

4301 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4302 {
4303   \if_int_compare:w #3 > #1
4304   \else:
4305     \if_int_compare:w #1 > #5
4306     \else:
4307       \if_int_compare:w #4 > #2
4308       \else:
4309         \if_int_compare:w #2 > #6
4310         \else:
4311           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4312           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4313           \if_int_compare:w \l_@@_row_max_int > #5 \l_@@_row_max_int = #5 \fi:
4314           \if_int_compare:w \l_@@_col_max_int > #6 \l_@@_col_max_int = #6 \fi:

```

```

4315         \fi:
4316     \fi:
4317     \fi:
4318     \fi:
4319 }

4320 \cs_new_protected:Npn \@@_set_initial_coords:
4321 {
4322     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4323     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4324 }
4325 \cs_new_protected:Npn \@@_set_final_coords:
4326 {
4327     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4328     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4329 }
4330 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4331 {
4332     \pgfpointanchor
4333     {
4334         \@@_env:
4335         - \int_use:N \l_@@_initial_i_int
4336         - \int_use:N \l_@@_initial_j_int
4337     }
4338     { #1 }
4339     \@@_set_initial_coords:
4340 }
4341 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4342 {
4343     \pgfpointanchor
4344     {
4345         \@@_env:
4346         - \int_use:N \l_@@_final_i_int
4347         - \int_use:N \l_@@_final_j_int
4348     }
4349     { #1 }
4350     \@@_set_final_coords:
4351 }

4352 \cs_new_protected:Npn \@@_open_x_initial_dim:
4353 {
4354     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4355     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4356     {
4357         \cs_if_exist:cT
4358         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4359         {
4360             \pgfpointanchor
4361             { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4362             { west }
4363             \dim_set:Nn \l_@@_x_initial_dim
4364             { \dim_min:nn { \l_@@_x_initial_dim } { \pgf@x } }
4365         }
4366     }

```

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

```

4367     \dim_compare:nNnT { \l_@@_x_initial_dim } = { \c_max_dim }
4368     {
4369         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4370         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4371         \dim_add:Nn \l_@@_x_initial_dim \col@sep
4372     }
4373 }

```

```

4374 \cs_new_protected:Npn \@@_open_x_final_dim:
4375 {
4376   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4377   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4378   {
4379     \cs_if_exist:cT
4380     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4381     {
4382       \pgfpointanchor
4383       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4384       { east }
4385       \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
4386       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4387     }
4388   }

```

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

```

4389   \dim_compare:nNnT { \l_@@_x_final_dim } = { - \c_max_dim }
4390   {
4391     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4392     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4393     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4394   }
4395 }

```

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.

```

4396 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4397 {
4398   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4399   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4400   {
4401     \@@_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.

```

4402   \bool_if:NT \g_@@_aux_found_bool
4403   {
4404     \group_begin:
4405     \@@_open_shorten:
4406     \int_if_zero:nTF { #1 }
4407     { \color { nicematrix-first-row } }
4408     {

```

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.

```

4409       \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4410       { \color { nicematrix-last-row } }
4411     }
4412     \keys_set:nn { nicematrix / xdots } { #3 }
4413     \@@_color:o \l_@@_xdots_color_tl
4414     \@@_actually_draw_Ldots:
4415   \group_end:
4416 }
4417 }
4418 }

```

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.

```

4419 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4420 {
4421   \bool_if:NTF \l_@@_initial_open_bool
4422   {
4423     \@@_open_x_initial_dim:
4424     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4425     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4426   }
4427   { \@@_set_initial_coords_from_anchor:n { base-east } }
4428   \bool_if:NTF \l_@@_final_open_bool
4429   {
4430     \@@_open_x_final_dim:
4431     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4432     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4433   }
4434   { \@@_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.

```

4435   \bool_lazy_all:nTF
4436   {
4437     \l_@@_initial_open_bool
4438     \l_@@_final_open_bool
4439     { \int_compare_p:nNn { \l_@@_initial_i_int } = { \l_@@_last_row_int } }
4440   }
4441   {
4442     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4443     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4444   }

```

We raise the line of a quantity equal to the radius of the dots because we want the dots really “on” the line of texte. Of course, maybe we should not do that when the option `line-style` is used (?).

```

4445   {
4446     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4447     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4448   }
4449   \@@_draw_line:
4450 }

```

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.

```

4451 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4452 {
4453   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4454   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4455   {
4456     \@@_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.

```

4457   \bool_if:NT \g_@@_aux_found_bool
4458   {
4459     \group_begin:
4460     \@@_open_shorten:
4461     \int_if_zero:nTF { #1 }
4462     { \color { nicematrix-first-row } }
4463     {

```

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.

```

4464         \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4465         { \color { nicematrix-last-row } }
4466     }
4467     \keys_set:nn { nicematrix / xdots } { #3 }
4468     \@@_color:o \l_@@_xdots_color_tl
4469     \@@_actually_draw_Cdots:
4470 \group_end:
4471 }
4472 }
4473 }

```

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.`

```

4474 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4475 {
4476     \bool_if:NTF \l_@@_initial_open_bool
4477     { \@@_open_x_initial_dim: }
4478     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4479     \bool_if:NTF \l_@@_final_open_bool
4480     { \@@_open_x_final_dim: }
4481     { \@@_set_final_coords_from_anchor:n { mid-west } }
4482     \bool_lazy_and:nnTF
4483     { \l_@@_initial_open_bool }
4484     { \l_@@_final_open_bool }
4485     {
4486         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4487         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4488         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4489         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4490         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4491     }
4492     {
4493         \bool_if:NT \l_@@_initial_open_bool
4494         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4495         \bool_if:NT \l_@@_final_open_bool
4496         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4497     }
4498     \@@_draw_line:
4499 }
4500 \cs_new_protected:Npn \@@_open_y_initial_dim:
4501 {
4502     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4503     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4504     {
4505         \cs_if_exist:cT
4506         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4507         {
4508             \pgfpointanchor
4509             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4510             { north }

```



```

4511         \dim_compare:nNnT { \pgf@y } > { \l_@@_y_initial_dim }
4512         { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4513     }
4514 }
4515 \dim_compare:nNnT { \l_@@_y_initial_dim } = { - \c_max_dim }
4516 {
4517     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4518     \dim_set:Nn \l_@@_y_initial_dim
4519     {
4520         \fp_to_dim:n
4521         {
4522             \pgf@y
4523             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4524         }
4525     }
4526 }
4527 }
4528 \cs_new_protected:Npn \@@_open_y_final_dim:
4529 {
4530     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4531     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4532     {
4533         \cs_if_exist:cT
4534         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4535         {
4536             \pgfpointanchor
4537             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4538             { south }
4539             \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
4540             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4541         }
4542     }
4543     \dim_compare:nNnT { \l_@@_y_final_dim } = { \c_max_dim }
4544     {
4545         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4546         \dim_set:Nn \l_@@_y_final_dim
4547         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4548     }
4549 }

```

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.

```

4550 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4551 {
4552     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4553     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4554     {
4555         \@@_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.

```

4556     \bool_if:NT \g_@@_aux_found_bool
4557     {
4558         \group_begin:
4559         \@@_open_shorten:
4560         \int_if_zero:nTF { #2 }
4561         { \color { nicematrix-first-col } }
4562         {
4563             \int_compare:nNnT { #2 } = { \l_@@_last_col_int }
4564             { \color { nicematrix-last-col } }
4565         }
4566         \keys_set:nn { nicematrix / xdots } { #3 }
4567         \@@_color:o \l_@@_xdots_color_tl
4568         \bool_if:NTF \l_@@_Vbrace_bool

```

```

4569         { \@@_actually_draw_Vbrace: }
4570         { \@@_actually_draw_Vdots: }
4571     \group_end:
4572 }
4573 }
4574 }

```

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.`

```

4575 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4576 {
4577     \bool_lazy_and:nnTF { \l_@@_initial_open_bool } { \l_@@_final_open_bool }
4578     { \@@_actually_draw_Vdots_i: }
4579     { \@@_actually_draw_Vdots_ii: }
4580     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4581     \@@_draw_line:
4582 }

```

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

```

4583 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4584 {
4585     \@@_open_y_initial_dim:
4586     \@@_open_y_final_dim:
4587     \int_if_zero:nTF { \l_@@_initial_j_int }

```

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

```

4588     {
4589         \@@_qpoint:n { col - 1 }
4590         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4591         \dim_sub:Nn \l_@@_x_initial_dim
4592         { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4593     }
4594     {
4595         \bool_lazy_and:nnTF
4596         { \int_compare_p:nNn { \l_@@_last_col_int } > { -2 } }
4597         {
4598             \int_compare_p:nNn
4599             { \l_@@_initial_j_int } = { \g_@@_col_total_int }
4600         }

```

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

```

4601     {
4602         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4603         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4604         \dim_add:Nn \l_@@_x_initial_dim
4605         { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4606     }

```

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

```

4607     {
4608         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4609         \dim_set_eq:NN \l_tmpa_dim \pgf@x
4610         \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4611         \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4612     }
4613 }
4614 }

```

The command `\@@_draw_line:` is in `\@@_actually_draw_Vdots:`

Now, the dotted line is *not* open on both sides (maybe open on only one side).

The main task is to determine the x -value of the dotted line to draw.

The boolean `\l_tmpa_bool` will indicate whether the column is of type `l` or may be considered as if.

```

4615 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4616 {
4617     \bool_set_false:N \l_tmpa_bool
4618     \bool_if:NF \l_@@_initial_open_bool
4619     {
4620         \bool_if:NF \l_@@_final_open_bool
4621         {
4622             \@@_set_initial_coords_from_anchor:n { south-west }
4623             \@@_set_final_coords_from_anchor:n { north-west }
4624             \bool_set:Nn \l_tmpa_bool
4625             {
4626                 \dim_compare_p:nNn
4627                 { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4628             }
4629         }
4630     }

```

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

```

4631     \bool_if:NTF \l_@@_initial_open_bool
4632     {
4633         \@@_open_y_initial_dim:
4634         \@@_set_final_coords_from_anchor:n { north }
4635         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4636     }
4637     {
4638         \@@_set_initial_coords_from_anchor:n { south }
4639         \bool_if:NTF \l_@@_final_open_bool
4640         { \@@_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.

```

4641     {
4642         \@@_set_final_coords_from_anchor:n { north }
4643         \dim_compare:nNnF { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4644         {
4645             \dim_set:Nn \l_@@_x_initial_dim
4646             {
4647                 \bool_if:NTF \l_tmpa_bool { \dim_min:nn } { \dim_max:nn }
4648                 \l_@@_x_initial_dim \l_@@_x_final_dim
4649             }
4650         }
4651     }
4652 }
4653 }

```

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`.

```

4654 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4655 {
4656   \bool_if:NTF \l_@@_initial_open_bool
4657     { \@@_open_y_initial_dim: }
4658     { \@@_set_initial_coords_from_anchor:n { south } }
4659   \bool_if:NTF \l_@@_final_open_bool
4660     { \@@_open_y_final_dim: }
4661     { \@@_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.

```

4662   \int_if_zero:nTF { \l_@@_initial_j_int }
4663   {
4664     \@@_qpoint:n { col - 1 }
4665     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4666     \dim_sub:Nn \l_@@_x_initial_dim
4667       { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4668   }

```

Elsewhere, the brace must be drawn left flush.

```

4669   {
4670     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4671     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4672     \dim_add:Nn \l_@@_x_initial_dim
4673       { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4674   }

```

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

```

4675   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4676   \@@_draw_line:
4677 }

```

```

4678 \cs_new:Npn \@@_colsep:
4679 { \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.

```

4680 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4681 {
4682   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4683   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4684   {
4685     \@@_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.

```

4686     \bool_if:NT \g_@@_aux_found_bool
4687     {
4688         \group_begin:
4689         \@@_open_shorten:
4690         \keys_set:nn { nicematrix / xdots } { #3 }
4691         \@@_color:o \l_@@_xdots_color_tl
4692         \@@_actually_draw_Ddots:
4693         \group_end:
4694     }
4695 }
4696 }
```

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`.

```

4697 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4698 {
4699     \bool_if:NTF \l_@@_initial_open_bool
4700     {
4701         \@@_open_y_initial_dim:
4702         \@@_open_x_initial_dim:
4703     }
4704     { \@@_set_initial_coords_from_anchor:n { south~east } }
4705     \bool_if:NTF \l_@@_final_open_bool
4706     {
4707         \@@_open_x_final_dim:
4708         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4709     }
4710     { \@@_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.

```

4711     \bool_if:NT \l_@@_parallelize_diags_bool
4712     {
4713         \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).

```

4714         \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.

```

4715     {
4716         \dim_gset:Nn \g_@@_delta_x_one_dim
4717         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4718         \dim_gset:Nn \g_@@_delta_y_one_dim
4719         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4720     }
```

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`.

```

4721     {
4722         \dim_compare:nNnF { \g_@@_delta_x_one_dim } = { \c_zero_dim }
4723         {
4724             \dim_set:Nn \l_@@_y_final_dim
4725             {
4726                 \l_@@_y_initial_dim +
4727                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4728                 \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4729             }
4730         }
4731     }
4732 }
4733 \@@_draw_line:
4734 }

```

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.

```

4735 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4736 {
4737     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4738     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4739     {
4740         \@@_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.

```

4741         \bool_if:NT \g_@@_aux_found_bool
4742         {
4743             \group_begin:
4744             \@@_open_shorten:
4745             \keys_set:nn { nicematrix / xdots } { #3 }
4746             \@@_color:o \l_@@_xdots_color_tl
4747             \@@_actually_draw_Iddots:
4748             \group_end:
4749         }
4750     }
4751 }

```

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`.

```

4752 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4753 {
4754     \bool_if:NTF \l_@@_initial_open_bool
4755     {
4756         \@@_open_y_initial_dim:
4757         \@@_open_x_initial_dim:
4758     }
4759     { \@@_set_initial_coords_from_anchor:n { south-west } }
4760     \bool_if:NTF \l_@@_final_open_bool

```

```

4761 {
4762   \l_@@_open_y_final_dim:
4763   \l_@@_open_x_final_dim:
4764 }
4765 { \l_@@_set_final_coords_from_anchor:n { north-east } }
4766 \bool_if:NT \l_@@_parallelize_diags_bool
4767 {
4768   \int_gincr:N \g_@@_iddots_int
4769   \int_compare:nNnTF { \g_@@_iddots_int } = { \c_one_int }
4770   {
4771     \dim_gset:Nn \g_@@_delta_x_two_dim
4772     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4773     \dim_gset:Nn \g_@@_delta_y_two_dim
4774     { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4775   }
4776   {
4777     \dim_compare:nNnF { \g_@@_delta_x_two_dim } = { \c_zero_dim }
4778     {
4779       \dim_set:Nn \l_@@_y_final_dim
4780       {
4781         \l_@@_y_initial_dim +
4782         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4783         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4784       }
4785     }
4786   }
4787 }
4788 \l_@@_draw_line:
4789 }

```

17 The actual instructions for drawing the dotted lines with Tikz

The command `\l_@@_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`

```

4790 \cs_new_protected:Npn \l_@@_draw_line:
4791 {
4792   \pgfrememberpicturepositiononpagetrue
4793   \pgf@relevantforpicturesizefalse
4794   \bool_lazy_or:nnTF
4795   { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4796   { \l_@@_dotted_bool }
4797   { \l_@@_draw_standard_dotted_line: }
4798   { \l_@@_draw_unstandard_dotted_line: }
4799 }

```

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.

```

4800 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4801 {
4802   \begin { scope }
4803   \@@_draw_unstandard_dotted_line:o
4804   { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4805 }

```

We have used the fact that, in PGF, un color name can be put directly in a list of options (that's why we have put directly `\l_@@_xdots_color_tl`).

The argument of `\@@_draw_unstandard_dotted_line:n` is, in fact, the list of options.

```

4806 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4807 {
4808   \@@_draw_unstandard_dotted_line:nooo
4809   { #1 }
4810   \l_@@_xdots_up_tl
4811   \l_@@_xdots_down_tl
4812   \l_@@_xdots_middle_tl
4813 }
4814 \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.

```

4815 \hook_gput_code:nnn { begindocument } { . }
4816 {
4817   \IfPackageLoadedT { tikz }
4818   {
4819     \tikzset
4820     {
4821       @@_node_above / .style = { sloped , above } ,
4822       @@_node_below / .style = { sloped , below } ,
4823       @@_node_middle / .style =
4824       {
4825         sloped ,
4826         inner~sep = \c_@@_innersep_middle_dim
4827       }
4828     }
4829   }
4830 }

4831 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4832 {

```

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.

```

4833 \dim_zero_new:N \l_@@_l_dim
4834 \dim_set:Nn \l_@@_l_dim
4835 {
4836   \fp_to_dim:n
4837   {
4838     sqrt
4839     (
4840       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4841       +
4842       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4843     )
4844   }
4845 }

```


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.

```

4846   \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4847   {
4848     \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4849     \@@_draw_unstandard_dotted_line_i:
4850   }

```

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

```

4851   \bool_if:NT \l_@@_xdots_h_labels_bool
4852   {
4853     \tikzset
4854     {
4855       @@_node_above / .style = { auto = left } ,
4856       @@_node_below / .style = { auto = right } ,
4857       @@_node_middle / .style = { innersep = \c_@@_innersep_middle_dim }
4858     }
4859   }
4860   \tl_if_empty:nF { #4 }
4861   { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4862   \dim_zero:N \l_tmpa_dim
4863   \dim_zero:N \l_tmpb_dim
4864   \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
4865   {

```

We test whether the brace is vertical or horizontal.

```

4866     \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4867     { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
4868     { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
4869   }
4870   {
4871     \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
4872     {
4873       \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4874       { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
4875       { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
4876     }
4877   }
4878   \use:e
4879   {
4880     \exp_not:N \begin { scope }
4881     [ shift = {(\dim_use:N \l_tmpa_dim,\dim_use:N \l_tmpb_dim)} ]
4882   }
4883   \draw
4884   [ #1 ]
4885   ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
4886   -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4887   node [ @@_node_below ] { $ \scriptstyle #3 $ }
4888   node [ @@_node_above ] { $ \scriptstyle #2 $ }
4889   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4890   \end { scope }
4891   \end { scope }
4892 }
4893 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4894 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4895 {
4896   \dim_set:Nn \l_tmpa_dim
4897   {
4898     \l_@@_x_initial_dim
4899     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4900     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim

```

```

4901     }
4902     \dim_set:Nn \l_tmpb_dim
4903     {
4904         \l_@@_y_initial_dim
4905         + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4906         * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4907     }
4908     \dim_set:Nn \l_@@_tmpc_dim
4909     {
4910         \l_@@_x_final_dim
4911         - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4912         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4913     }
4914     \dim_set:Nn \l_@@_tmpd_dim
4915     {
4916         \l_@@_y_final_dim
4917         - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4918         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4919     }
4920     \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4921     \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4922     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4923     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4924 }

```

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

```

4925 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4926 {
4927     \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.

```

4928     \dim_zero_new:N \l_@@_l_dim
4929     \dim_set:Nn \l_@@_l_dim
4930     {
4931         \fp_to_dim:n
4932         {
4933             sqrt
4934             (
4935                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4936                 +
4937                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4938             )
4939         }
4940     }

```

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.

```

4941     \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4942     {
4943         \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4944         { \@@_draw_standard_dotted_line_i: }
4945     }
4946     \group_end:
4947     \bool_lazy_all:nF
4948     {
4949         { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4950         { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4951         { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4952     }

```

```

4953     { \l_labels_standard_dotted_line: }
4954 }
4955 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4956 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4957 {

```

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

```

4958     \int_set:Nn \l_tmpa_int
4959     {
4960         \dim_ratio:nn
4961         {
4962             \l_@@_l_dim
4963             - \l_@@_xdots_shorten_start_dim
4964             - \l_@@_xdots_shorten_end_dim
4965         }
4966         { \l_@@_xdots_inter_dim }
4967     }

```

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

```

4968     \dim_set:Nn \l_tmpa_dim
4969     {
4970         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4971         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4972     }
4973     \dim_set:Nn \l_tmpb_dim
4974     {
4975         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4976         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4977     }

```

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.

```

4978     \dim_gadd:Nn \l_@@_x_initial_dim
4979     {
4980         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4981         \dim_ratio:nn
4982         {
4983             \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4984             + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4985         }
4986         { 2 \l_@@_l_dim }
4987     }
4988     \dim_gadd:Nn \l_@@_y_initial_dim
4989     {
4990         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4991         \dim_ratio:nn
4992         {
4993             \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4994             + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4995         }
4996         { 2 \l_@@_l_dim }
4997     }
4998     \pgf@relevantforpicturesizefalse
4999     \int_step_inline:nnn { \c_zero_int } { \l_tmpa_int }
5000     {
5001         \pgfpathcircle
5002         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5003         { \l_@@_xdots_radius_dim }
5004         \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
5005         \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
5006     }
5007     \pgfusepathqfill
5008 }

```

```

5009 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
5010 {
5011   \pgfscope
5012   \pgftransformshift
5013   {
5014     \pgfpointlineattime { 0.5 }
5015     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5016     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
5017   }
5018   \fp_set:Nn \l_tmpa_fp
5019   {
5020     atand
5021     (
5022       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
5023       \l_@@_x_final_dim - \l_@@_x_initial_dim
5024     )
5025   }
5026   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
5027   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
5028   \tl_if_empty:NF \l_@@_xdots_middle_tl
5029   {
5030     \begin { pgfscope }
5031     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
5032     \pgfnode
5033     { rectangle }
5034     { center }
5035     {
5036       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5037       {
5038         $ % $
5039         \scriptstyle \l_@@_xdots_middle_tl
5040         $ % $
5041       }
5042     }
5043     { }
5044     {
5045       \pgfsetfillcolor { white }
5046       \pgfusepath { fill }
5047     }
5048     \end { pgfscope }
5049   }
5050   \tl_if_empty:NF \l_@@_xdots_up_tl
5051   {
5052     \pgfnode
5053     { rectangle }
5054     { south }
5055     {
5056       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5057       {
5058         $ % $
5059         \scriptstyle \l_@@_xdots_up_tl
5060         $ % $
5061       }
5062     }
5063     { }
5064     { \pgfusepath { } }
5065   }
5066   \tl_if_empty:NF \l_@@_xdots_down_tl
5067   {
5068     \pgfnode
5069     { rectangle }
5070     { north }
5071     {

```

```

5072         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5073         {
5074             $ % $
5075             \scriptstyle \l_@@_xdots_down_tl
5076             $ % $
5077         }
5078     }
5079     { }
5080     { \pgfusepath { } }
5081 }
5082 \endpgfscope
5083 }

```

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.

```

5084 \hook_gput_code:nnn { begindocument } { . }
5085 {

```

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

```

5086 \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
5087 \cs_new_protected:Npn \@@_Ldots:
5088 { \@@_collect_options:n { \@@_Ldots_i } }
5089 \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5090 {
5091     \int_if_zero:nTF { \c@jCol }
5092     { \@@_error:nn { in~first~col } { \Ldots } }
5093     {
5094         \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5095         { \@@_error:nn { in~last~col } { \Ldots } }
5096         {
5097             \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5098             { #1 , down = #2 , up = #3 , middle = #4 }
5099         }
5100     }
5101     \bool_if:NF \l_@@_nullify_dots_bool
5102     { \phantom { \ensuremath { \@@_old_ldots: } } }
5103     \bool_gset_true:N \g_@@_empty_cell_bool
5104 }

5105 \cs_new_protected:Npn \@@_Cdots:
5106 { \@@_collect_options:n { \@@_Cdots_i } }
5107 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5108 {
5109     \int_if_zero:nTF { \c@jCol }
5110     { \@@_error:nn { in~first~col } { \Cdots } }
5111     {
5112         \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5113         { \@@_error:nn { in~last~col } { \Cdots } }

```

```

5114         {
5115             \@@_instruction_of_type:nnn { \c_false_bool } { Cdots }
5116             { #1 , down = #2 , up = #3 , middle = #4 }
5117         }
5118     }
5119     \bool_if:NF \l_@@_nullify_dots_bool
5120     { \phantom { \ensuremath { \@@_old_cdots: } } }
5121     \bool_gset_true:N \g_@@_empty_cell_bool
5122 }

5123 \cs_new_protected:Npn \@@_Vdots:
5124 { \@@_collect_options:n { \@@_Vdots_i } }
5125 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5126 {
5127     \int_if_zero:nTF { \c@iRow }
5128     { \@@_error:nn { in~first~row } { \Vdots } }
5129     {
5130         \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
5131         { \@@_error:nn { in~last~row } { \Vdots } }
5132         {
5133             \@@_instruction_of_type:nnn { \c_false_bool } { Vdots }
5134             { #1 , down = #2 , up = #3 , middle = #4 }
5135         }
5136     }
5137     \bool_if:NF \l_@@_nullify_dots_bool
5138     { \phantom { \ensuremath { \@@_old_vdots: } } }
5139     \bool_gset_true:N \g_@@_empty_cell_bool
5140 }

5141 \cs_new_protected:Npn \@@_Ddots:
5142 { \@@_collect_options:n { \@@_Ddots_i } }
5143 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5144 {
5145     \int_case:nnF \c@iRow
5146     {
5147         0 { \@@_error:nn { in~first~row } { \Ddots } }
5148         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5149     }
5150     {
5151         \int_case:nnF \c@jCol
5152         {
5153             0 { \@@_error:nn { in~first~col } { \Ddots } }
5154             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5155         }
5156         {
5157             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5158             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5159             { #1 , down = #2 , up = #3 , middle = #4 }
5160         }
5161     }
5162 }
5163 \bool_if:NF \l_@@_nullify_dots_bool
5164 { \phantom { \ensuremath { \@@_old_ddots: } } }
5165 \bool_gset_true:N \g_@@_empty_cell_bool
5166 }

5167 \cs_new_protected:Npn \@@_Iddots:
5168 { \@@_collect_options:n { \@@_Iddots_i } }
5169 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5170 {
5171     \int_case:nnF \c@iRow

```

```

5172     {
5173         0 { \@@_error:nn { in~first~row } { \Iddots } }
5174         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5175     }
5176     {
5177         \int_case:nnF \c@jCol
5178         {
5179             0 { \@@_error:nn { in~first~col } { \Iddots } }
5180             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5181         }
5182         {
5183             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5184             \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { Iddots }
5185             { #1 , down = #2 , up = #3 , middle = #4 }
5186         }
5187     }
5188     \bool_if:NF \l_@@_nullify_dots_bool
5189     { \phantom { \ensuremath { \@@_old_iddots: } } }
5190     \bool_gset_true:N \g_@@_empty_cell_bool
5191 }
5192 }

```

End of the `\AddToHook`.

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

```

5193 \keys_define:nn { nicematrix / Ddots }
5194 {
5195     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5196     draw-first .default:n = true ,
5197     draw-first .value_forbidden:n = true
5198 }

```

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

```

5199 \cs_new_protected:Npn \@@_Hspace:
5200 {
5201     \bool_gset_true:N \g_@@_empty_cell_bool
5202     \hspace
5203 }

```

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`.

```

5204 \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`.

```

5205 \cs_new:Npn \@@_Hdotsfor:
5206 {
5207     \bool_lazy_and:nnTF
5208     { \int_if_zero_p:n { \c@jCol } }
5209     { \int_if_zero_p:n { \l_@@_first_col_int } }
5210     {
5211         \bool_if:NTF \g_@@_after_col_zero_bool
5212         {
5213             \multicolumn { 1 } { c } { }
5214             \@@_Hdotsfor_i:
5215         }
5216         { \@@_fatal:n { Hdotsfor~in~col~0 } }
5217     }
5218     {
5219         \multicolumn { 1 } { c } { }

```

```

5220 \@@_Hdotsfor_i:
5221 }
5222 }

```

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:`).

```

5223 \hook_gput_code:nnn { begindocument } { . }
5224 {

```

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

```

5225 \cs_new_protected:Npn \@@_Hdotsfor_i:
5226 { \@@_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`).

```

5227 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m O { } E { _ ^ : } { { } { } { } } }
5228 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5229 {
5230 \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5231 {
5232 \@@_Hdotsfor:nnnn
5233 { \int_use:N \c@iRow }
5234 { \int_use:N \c@jCol }
5235 { #2 }
5236 {
5237 #1 , #3 ,
5238 down = \exp_not:n { #4 } ,
5239 up = \exp_not:n { #5 } ,
5240 middle = \exp_not:n { #6 }
5241 }
5242 }
5243 \prg_replicate:nn { #2 - 1 }
5244 {
5245 &
5246 \multicolumn { 1 } { c } { }
5247 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5248 }
5249 }
5250 }

```

```

5251 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5252 {
5253 \bool_set_false:N \l_@@_initial_open_bool
5254 \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5255 \int_set:Nn \l_@@_initial_i_int { #1 }
5256 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5257 \int_compare:nNnTF { #2 } = { \c_one_int }
5258 {
5259 \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5260 \bool_set_true:N \l_@@_initial_open_bool
5261 }
5262 {
5263 \cs_if_exist:cTF
5264 {
5265 pgf @ sh @ ns @ \@@_env:
5266 - \int_use:N \l_@@_initial_i_int
5267 - \int_eval:n { #2 - 1 }
5268 }
5269 { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }

```



```

5270     {
5271         \int_set:Nn \l_@@_initial_j_int { #2 }
5272         \bool_set_true:N \l_@@_initial_open_bool
5273     }
5274 }
5275 \int_compare:nNnTF { #2 + #3 - 1 } = { \c@jCol }
5276 {
5277     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5278     \bool_set_true:N \l_@@_final_open_bool
5279 }
5280 {
5281     \cs_if_exist:cTF
5282     {
5283         pgf @ sh @ ns @ \@@_env:
5284         - \int_use:N \l_@@_final_i_int
5285         - \int_eval:n { #2 + #3 }
5286     }
5287     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5288     {
5289         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5290         \bool_set_true:N \l_@@_final_open_bool
5291     }
5292 }
5293 \bool_if:NT \g_@@_aux_found_bool
5294 {
5295     \group_begin:
5296     \@@_open_shorten:
5297     \int_if_zero:nTF { #1 }
5298     { \color { nicematrix-first-row } }
5299     {
5300         \int_compare:nNnT { #1 } = { \g_@@_row_total_int }
5301         { \color { nicematrix-last-row } }
5302     }
5303     \keys_set:nn { nicematrix / xdots } { #4 }
5304     \@@_color:o \l_@@_xdots_color_tl
5305     \@@_actually_draw_Ldots:
5306     \group_end:
5307 }

```

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).

```

5308     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5309     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5310 }

```

```

5311 \hook_gput_code:nnn { begindocument } { . }
5312 {
5313     \cs_new_protected:Npn \@@_Vdotsfor:
5314     { \@@_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`).

```

5315     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5316     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5317     {
5318         \bool_gset_true:N \g_@@_empty_cell_bool
5319         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5320         {
5321             \@@_Vdotsfor:nnnn
5322             { \int_use:N \c@iRow }
5323             { \int_use:N \c@jCol }

```

```

5324         { #2 }
5325     {
5326         #1 , #3 ,
5327         down = \exp_not:n { #4 } ,
5328         up = \exp_not:n { #5 } ,
5329         middle = \exp_not:n { #6 }
5330     }
5331 }
5332 }
5333 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5334 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5335 {
5336     \bool_set_false:N \l_@@_initial_open_bool
5337     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5338     \int_set:Nn \l_@@_initial_j_int { #2 }
5339     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5340     \int_compare:nNnTF { #1 } = { \c_one_int }
5341     {
5342         \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5343         \bool_set_true:N \l_@@_initial_open_bool
5344     }
5345     {
5346         \cs_if_exist:cTF
5347         {
5348             pgf @ sh @ ns @ \@@_env:
5349             - \int_eval:n { #1 - 1 }
5350             - \int_use:N \l_@@_initial_j_int
5351         }
5352         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5353         {
5354             \int_set:Nn \l_@@_initial_i_int { #1 }
5355             \bool_set_true:N \l_@@_initial_open_bool
5356         }
5357     }
5358     \int_compare:nNnTF { #1 + #3 - 1 } = { \c@iRow }
5359     {
5360         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5361         \bool_set_true:N \l_@@_final_open_bool
5362     }
5363     {
5364         \cs_if_exist:cTF
5365         {
5366             pgf @ sh @ ns @ \@@_env:
5367             - \int_eval:n { #1 + #3 }
5368             - \int_use:N \l_@@_final_j_int
5369         }
5370         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5371         {
5372             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5373             \bool_set_true:N \l_@@_final_open_bool
5374         }
5375     }
5376     \bool_if:NT \g_@@_aux_found_bool
5377     {
5378         \group_begin:

```

```

5379 \@@_open_shorten:
5380 \int_if_zero:nTF { #2 }
5381 { \color { nicematrix-first-col } }
5382 {
5383 \int_compare:nNtT { #2 } = { \g_@@_col_total_int }
5384 { \color { nicematrix-last-col } }
5385 }
5386 \keys_set:nn { nicematrix / xdots } { #4 }
5387 \@@_color:o \l_@@_xdots_color_tl
5388 \bool_if:NTF \l_@@_Vbrace_bool
5389 { \@@_actually_draw_Vbrace: }
5390 { \@@_actually_draw_Vdots: }
5391 \group_end:
5392 }

```

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).

```

5393 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5394 { \cs_set_nopar:cpn { @@_dotted_ ##1 - #2 } { } }
5395 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5396 \NewDocumentCommand \@@_rotate: { 0 { } }
5397 {
5398 \bool_gset_true:N \g_@@_rotate_bool
5399 \keys_set:nn { nicematrix / rotate } { #1 }
5400 \ignorespaces
5401 }

5402 \keys_define:nn { nicematrix / rotate }
5403 {
5404 c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5405 c .value_forbidden:n = true ,
5406 unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5407 }

```

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

```

5408 \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`.

```

5409 {
5410   \tl_if_empty:nTF { #2 }
5411     { #1 }
5412     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5413 }
5414 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5415 { \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).

```

5416 \hook_gput_code:nnn { begindocument } { . }
5417 {

```

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

```

5418   \tl_set_rescan:Nnn \l_tmpa_tl { }
5419   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5420   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5421     {
5422       \group_begin:
5423       \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5424       \@@_color:o \l_@@_xdots_color_tl
5425       \use:e
5426       {
5427         \@@_line_i:nn
5428         { \@@_double_int_eval:n #2 - \q_stop }
5429         { \@@_double_int_eval:n #3 - \q_stop }
5430       }
5431       \group_end:
5432     }
5433 }
5434 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5435 {
5436   \bool_set_false:N \l_@@_initial_open_bool
5437   \bool_set_false:N \l_@@_final_open_bool
5438   \bool_lazy_or:nnTF
5439     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5440     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5441     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

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

```

5442   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5443 }
5444 \hook_gput_code:nnn { begindocument } { . }
5445 {
5446   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5447   {

```

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:`.

```

5448     \c_@@_pgfortikzpicture_tl
5449     \@@_draw_line_iii:nn { #1 } { #2 }
5450     \c_@@_endpgfortikzpicture_tl
5451   }
5452 }

```

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

```

5453 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5454 {
5455   \pgfrememberpicturepositiononpagetrue

```

```

5456 \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5457 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5458 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5459 \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5460 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5461 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5462 \@@_draw_line:
5463 }

```

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.

```

5464 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNnT { \c@iRow } < }
5465 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNnF { \c@jCol } < }

```

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

```

5466 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5467 {
5468   \tl_gput_right:Ne \g_@@_row_style_tl
5469   {

```

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

```

5470 \exp_not:N
5471 \@@_if_row_less_than:nn
5472 { \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`).

```

5473 {
5474   \exp_not:N
5475   \@@_if_col_greater_than:nn
5476   { \int_eval:n { \c@jCol } }
5477   { \exp_not:n { #1 } \scan_stop: }
5478 }
5479 }
5480 }
5481 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

```

```

5482 \keys_define:nn { nicematrix / RowStyle }
5483 {
5484   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5485   cell-space-top-limit .value_required:n = true ,
5486   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5487   cell-space-bottom-limit .value_required:n = true ,

```

```

5488 cell-space-limits .meta:n =
5489 {
5490     cell-space-top-limit = #1 ,
5491     cell-space-bottom-limit = #1 ,
5492 } ,
5493 color .tl_set:N = \l_@@_color_tl ,
5494 color .value_required:n = true ,
5495 bold .bool_set:N = \l_@@_bold_row_style_bool ,
5496 bold .default:n = true ,
5497 nb-rows .code:n =
5498     \str_if_eq:eeTF { #1 } { * }
5499     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5500     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5501 nb-rows .value_required:n = true ,
5502 fill .tl_set:N = \l_@@_fill_tl ,
5503 fill .value_required:n = true ,

```

In fine, the opacity will be applied by `\pgfsetfillopacity`.

```

5504 opacity .tl_set:N = \l_@@_opacity_tl ,
5505 opacity .value_required:n = true ,
5506 rowcolor .tl_set:N = \l_@@_fill_tl ,
5507 rowcolor .value_required:n = true ,
5508 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5509 rounded-corners .default:n = 4 pt ,
5510 unknown .code:n =
5511     \@@_unknown_key:nn
5512     { nicematrix / RowStyle }
5513     { Unknown-key-for-RowStyle }
5514 }

```

```

5515 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5516 {
5517     \group_begin:
5518     \tl_clear:N \l_@@_fill_tl
5519     \tl_clear:N \l_@@_opacity_tl
5520     \tl_clear:N \l_@@_color_tl
5521     \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5522     \dim_zero:N \l_@@_rounded_corners_dim
5523     \dim_zero:N \l_tmpa_dim
5524     \dim_zero:N \l_tmpb_dim
5525     \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5526     \tl_if_empty:NF \l_@@_fill_tl
5527     {
5528         \@@_add_opacity_to_fill:
5529         \tl_gput_right:Ne \g_@@_pre_code_before_tl
5530         {

```

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

```

5531         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5532         { \int_use:N \c_iRow - \int_use:N \c_jCol }
5533         {
5534             \int_eval:n { \c_iRow + \l_@@_key_nb_rows_int - 1 }
5535             - *
5536         }
5537         { \dim_use:N \l_@@_rounded_corners_dim }
5538     }
5539 }
5540 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5541     \dim_compare:nNnT { \l_tmpa_dim } > { \c_zero_dim }
5542     {

```

```

5543 \@@_put_in_row_style:e
5544 {
5545     \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5546     {

```

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

```

5547         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5548         { \dim_use:N \l_tmpa_dim }
5549     }
5550 }
5551 }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

5552 \dim_compare:nNnT { \l_tmpb_dim } > { \c_zero_dim }
5553 {
5554     \@@_put_in_row_style:e
5555     {
5556         \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5557         {
5558             \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5559             { \dim_use:N \l_tmpb_dim }
5560         }
5561     }
5562 }

```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

5563 \tl_if_empty:NF \l_@@_color_tl
5564 {
5565     \@@_put_in_row_style:e
5566     {
5567         \mode_leave_vertical:
5568         \@@_color:n { \l_@@_color_tl }
5569     }
5570 }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5571 \bool_if:NT \l_@@_bold_row_style_bool
5572 {
5573     \@@_put_in_row_style:n
5574     {
5575         \exp_not:n
5576         {
5577             \if_mode_math:
5578             $ % $
5579             \bfseries \boldmath
5580             $ % $
5581             \else:
5582             \bfseries \boldmath
5583             \fi:
5584         }
5585     }
5586 }
5587 \group_end:
5588 \g_@@_row_style_tl
5589 \ignorespaces
5590 }

```

The following commande must *not* be protected.

```

5591 \cs_new:Npn \@@_rounded_from_row:n #1
5592 {
5593     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

In the following code, the “`- 1`” is *not* a subtraction.

```

5594 { \int_eval:n { #1 } - 1 }
5595 {

```

```

5596      \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5597      - \exp_not:n { \int_use:N \c@jCol }
5598    }
5599    { \dim_use:N \l_@@_rounded_corners_dim }
5600  }

```

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).

```

5601 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5602 {

```

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.

```

5603   \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`.

```

5604   \str_if_in:nnF { #1 } { !! }
5605   {
5606     \seq_map_indexed_inline:Nn \g_@@_colors_seq

```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```

5607     { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5608   }
5609   \int_if_zero:nTF { \l_tmpa_int }

```

First, the case where the color is a *new* color (not in the sequence).

```

5610   {
5611     \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5612     \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5613   }

```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```

5614   { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5615 }
5616 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5617 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }

```


The following command must be used within a `\pgfpicture`.

```

5618 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5619 {
5620   \dim_compare:nNtT { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim }
5621   {

```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5622   \group_begin:
5623   \pgfsetcornersarced
5624   {
5625     \pgfpoint
5626     { \l_@@_tab_rounded_corners_dim }
5627     { \l_@@_tab_rounded_corners_dim }
5628   }

```

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.

```

5629   \bool_if:NTF \l_@@_hvlines_bool
5630   {
5631     \pgfpathrectanglecorners
5632     {
5633       \pgfpointadd
5634       { \@@_qpoint:n { row-1 } }
5635       { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5636     }
5637     {
5638       \pgfpointadd
5639       {
5640         \@@_qpoint:n
5641         { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5642       }
5643       { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5644     }
5645   }
5646   {
5647     \pgfpathrectanglecorners
5648     { \@@_qpoint:n { row-1 } }
5649     {
5650       \pgfpointadd
5651       {
5652         \@@_qpoint:n
5653         { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5654       }
5655       { \pgfpoint \c_zero_dim \arrayrulewidth }
5656     }
5657   }
5658   \pgfusepath { clip }
5659   \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5660   }
5661 }

```

The macro `\@@_actually_color:` will actually fill all the rectangles, color by color (using the sequence `\l_@@_colors_seq` and all the token lists of the form `\l_@@_color-i-t1`).

```

5662 \cs_new_protected:Npn \@@_actually_color:
5663 {
5664   \pgfpicture
5665   \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.

```

5666 \@@_clip_with_rounded_corners:
5667 \seq_map_indexed_inline:Nn \g_@@_colors_seq
5668 {
5669   \int_compare:nNnTF { ##1 } = { \c_one_int }
5670   {
5671     \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5672     \use:c { g_@@_color _ 1 _tl }
5673     \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5674   }
5675   {
5676     \begin { pgfscope }
5677       \@@_color_opacity: ##2
5678       \use:c { g_@@_color _ ##1 _tl }
5679       \tl_gclear:c { g_@@_color _ ##1 _tl }
5680       \pgfusepath { fill }
5681     \end { pgfscope }
5682   }
5683 }
5684 \endpgfpicture
5685 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5686 \cs_new_protected:Npn \@@_color_opacity:
5687 {
5688   \peek_meaning:NTF [
5689     { \@@_color_opacity:w }
5690     { \@@_color_opacity:w [ ] }
5691   }

```

The command `\@@_color_opacity:w` takes in as argument only the optional argument. One may consider that the second argument (the actual definition of the color) is provided by curryfication.

```

5692 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5693 {
5694   \tl_clear:N \l_tmpa_tl
5695   \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.

```

5696   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5697   \tl_if_empty:NTF \l_tmpb_tl
5698     { \@declaredcolor }
5699     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5700 }

```

The following set of keys is used by the command `\@@_color_opacity:w`.

```

5701 \keys_define:nn { nicematrix / color-opacity }
5702 {
5703   opacity .tl_set:N          = \l_tmpa_tl ,
5704   opacity .value_required:n = true
5705 }

```

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

```

5706 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5707 {
5708   \def \l_@@_rows_tl { #1 }
5709   \def \l_@@_cols_tl { #2 }
5710   \@@_cartesian_path:
5711 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```
5712 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5713 {
5714   \tl_if_blank:nF { #2 }
5715   {
5716     \@@_add_to_colors_seq:en
5717     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5718     { \@@_cartesian_color:nn { #3 } { - } }
5719   }
5720 }
```

Here an example: `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```
5721 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5722 {
5723   \tl_if_blank:nF { #2 }
5724   {
5725     \@@_add_to_colors_seq:en
5726     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5727     { \@@_cartesian_color:nn { - } { #3 } }
5728   }
5729 }
```

Here is an example: `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```
5730 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5731 {
5732   \tl_if_blank:nF { #2 }
5733   {
5734     \@@_add_to_colors_seq:en
5735     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5736     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5737   }
5738 }
```

The last argument is the radius of the corners of the rectangle.

```
5739 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5740 {
5741   \tl_if_blank:nF { #2 }
5742   {
5743     \@@_add_to_colors_seq:en
5744     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5745     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5746   }
5747 }
```

The last argument is the radius of the corners of the rectangle.

```
5748 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5749 {
5750   \@@_cut_on_hyphen:w #1 \q_stop
5751   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5752   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5753   \@@_cut_on_hyphen:w #2 \q_stop
5754   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5755   \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`.

```
5756 \@@_cartesian_path:n { #3 }
5757 }
```

Here is an example: `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5758 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5759 {
5760   \clist_map_inline:nn { #3 }
5761     { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5762 }

5763 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5764 {
5765   \int_step_inline:nn { \c@iRow }
5766   {
5767     \int_step_inline:nn { \c@jCol }
5768     {
5769       \int_if_even:nTF { #####1 + ##1 }
5770         { \@@_cellcolor [ #1 ] { #2 } }
5771         { \@@_cellcolor [ #1 ] { #3 } }
5772       { ##1 - #####1 }
5773     }
5774   }
5775 }

```

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”.

```

5776 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5777 {
5778   \@@_rectanglecolor [ #1 ] { #2 }
5779   { 1 - 1 }
5780   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5781 }

5782 \keys_define:nn { nicematrix / rowcolors }
5783 {
5784   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5785   respect-blocks .default:n = true ,
5786   cols .tl_set:N = \l_@@_cols_tl ,
5787   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5788   restart .default:n = true ,
5789   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5790 }

```

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*.

```

5791 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5792 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5793   \group_begin:
5794   \seq_clear_new:N \l_@@_colors_seq
5795   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5796   \tl_clear_new:N \l_@@_cols_tl
5797   \tl_set:Nn \l_@@_cols_tl { - }
5798   \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).

```

5799     \int_zero_new:N \l_@@_color_int
5800     \int_set_eq:NN \l_@@_color_int \c_one_int
5801     \bool_if:NT \l_@@_respect_blocks_bool
5802     {

```

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`).

```

5803         \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5804         \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5805         { \@@_not_in_exterior_p:nnnnn ##1 }
5806     }
5807     \pgfpicture
5808     \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5809     \clist_map_inline:nn { #2 }
5810     {
5811         \tl_set:Nn \l_tmpa_tl { ##1 }
5812         \tl_if_in:NnTF \l_tmpa_tl { - }
5813         { \@@_cut_on_hyphen:w ##1 \q_stop }
5814         { \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.

```

5815         \int_set:Nn \l_tmpa_int \l_tmpa_tl
5816         \int_set:Nn \l_@@_color_int
5817         { \bool_if:NNTF \l_@@_rowcolors_restart_bool { 1 } { \l_tmpa_tl } }
5818         \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5819         \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5820         {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5821             \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).

```

5822             \bool_if:NT \l_@@_respect_blocks_bool
5823             {
5824                 \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5825                 { \@@_intersect_our_row_p:nnnnn #####1 }
5826                 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5827             }
5828             \tl_set:Nn \l_@@_rows_tl
5829             { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5830             \tl_set:Nn \l_@@_color_tl
5831             {
5832                 \@@_color_index:n
5833                 {
5834                     \int_mod:nn
5835                     { \l_@@_color_int - 1 }
5836                     { \seq_count:N \l_@@_colors_seq }
5837                     + 1
5838                 }
5839             }
5840             \tl_if_empty:NF \l_@@_color_tl
5841             {
5842                 \@@_add_to_colors_seq:ee
5843                 { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5844                 { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5845             }
5846             \int_incr:N \l_@@_color_int

```

```

5847         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5848     }
5849 }
5850 \endpgfpicture
5851 \group_end:
5852 }

```

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.

```

5853 \cs_new:Npn \@@_color_index:n #1
5854 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5855     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5856     { \@@_color_index:n { #1 - 1 } }
5857     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5858 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by curryfication.

```

5859 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5860 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5861 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5862 {
5863     \int_compare:nNnT { #3 } > { \l_tmpb_int }
5864     { \int_set:Nn \l_tmpb_int { #3 } }
5865 }

```

```

5866 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5867 {
5868     \int_if_zero:nTF { #4 }
5869     { \prg_return_false: }
5870     {
5871         \int_compare:nNnTF { #2 } > { \c@jCol }
5872         { \prg_return_false: }
5873         { \prg_return_true: }
5874     }
5875 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5876 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5877 {
5878     \int_compare:nNnTF { #1 } > { \l_tmpa_int }
5879     { \prg_return_false: }
5880     {
5881         \int_compare:nNnTF { \l_tmpa_int } > { #3 }
5882         { \prg_return_false: }
5883         { \prg_return_true: }
5884     }
5885 }

```

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`).

```

5886 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5887 {
5888   \dim_compare:nNnTF { #1 } = { \c_zero_dim }
5889   {
5890     \bool_if:NTF \l_@@_nocolor_used_bool
5891     { \@@_cartesian_path_normal_ii: }
5892     {
5893       \clist_if_empty:NTF \l_@@_corners_cells_clist
5894       { \@@_cartesian_path_normal_i:n { #1 } }
5895       { \@@_cartesian_path_normal_ii: }
5896     }
5897   }
5898   { \@@_cartesian_path_normal_i:n { #1 } }
5899 }

```

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.

```

5900 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5901 {
5902   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5903   \clist_map_inline:Nn \l_@@_cols_tl
5904   {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

5905     \def \l_tmpa_tl { ##1 }
5906     \tl_if_in:NnTF \l_tmpa_tl { - }
5907     { \@@_cut_on_hyphen:w ##1 \q_stop }
5908     { \def \l_tmpb_tl { ##1 } }
5909     \tl_if_empty:NTF \l_tmpa_tl
5910     { \def \l_tmpa_tl { 1 } }
5911     {
5912       \str_if_eq:eeT { \l_tmpa_tl } { * }
5913       { \def \l_tmpa_tl { 1 } }
5914     }
5915     \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_col_total_int }
5916     { \@@_error:n { Invalid~col~number } }
5917     \tl_if_empty:NTF \l_tmpb_tl
5918     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5919     {
5920       \str_if_eq:eeT { \l_tmpb_tl } { * }
5921       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5922     }
5923     \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_col_total_int }
5924     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5925     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5926     \@@_qpoint:n { col - \l_tmpa_tl }
5927     \int_compare:nNnTF { \l_@@_first_col_int } = { \l_tmpa_tl }
5928     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5929     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5930     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5931     \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.

```

5932     \clist_map_inline:Nn \l_@@_rows_tl
5933     {
5934       \def \l_tmpa_tl { ####1 }
5935       \tl_if_in:NnTF \l_tmpa_tl { - }
5936       { \@@_cut_on_hyphen:w ####1 \q_stop }

```

```

5937     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5938 \tl_if_empty:NTF \l_tmpa_tl
5939   { \def \l_tmpa_tl { 1 } }
5940   {
5941     \str_if_eq:eeT { \l_tmpa_tl } { * }
5942     { \def \l_tmpa_tl { 1 } }
5943   }
5944 \tl_if_empty:NTF \l_tmpb_tl
5945   { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5946   {
5947     \str_if_eq:eeT { \l_tmpb_tl } { * }
5948     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5949   }
5950 \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_row_total_int }
5951   { \@@_error:n { Invalid~row~number } }
5952 \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_row_total_int }
5953   { \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`.

```

5954 \cs_if_exist:cF
5955   { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5956   {
5957     \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5958     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5959     \@@_qpoint:n { row - \l_tmpa_tl }
5960     \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5961     \pgfpathrectanglecorners
5962       { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5963       { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5964   }
5965 }
5966 }
5967 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5968 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5969   {
5970     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5971     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5972 \clist_map_inline:Nn \l_@@_cols_tl
5973   {
5974     \@@_qpoint:n { col - ##1 }
5975     \int_compare:nNnTF { \l_@@_first_col_int } = { ##1 }
5976       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5977       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5978     \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5979     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5980 \clist_map_inline:Nn \l_@@_rows_tl
5981   {
5982     \@@_if_in_corner:nF { #####1 - ##1 }
5983     {
5984       \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5985       \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5986       \@@_qpoint:n { row - #####1 }
5987       \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5988       \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5989         {
5990           \pgfpathrectanglecorners
5991             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5992             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }

```



```

5993     }
5994   }
5995 }
5996 }
5997 }

```

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`).

```

5998 \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

```

5999 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
6000 {
6001   \bool_set_true:N \l_@@_nocolor_used_bool
6002   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6003   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6004   \clist_map_inline:Nn \l_@@_rows_tl
6005   {
6006     \clist_map_inline:Nn \l_@@_cols_tl
6007     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - ####1 } { } }
6008   }
6009 }

```

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`.

```

6010 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
6011 {
6012   \clist_set_eq:NN \l_tmpa_clist #1
6013   \clist_clear:N #1
6014   \clist_map_inline:Nn \l_tmpa_clist
6015   {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

6016     \def \l_tmpa_tl { ##1 }
6017     \tl_if_in:NnTF \l_tmpa_tl { - }
6018     { \@@_cut_on_hyphen:w ##1 \q_stop }
6019     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
6020     \bool_lazy_or:nnT
6021     { \str_if_eq_p:ee { \l_tmpa_tl } { * } }
6022     { \tl_if_blank_p:o \l_tmpa_tl }
6023     { \def \l_tmpa_tl { 1 } }
6024     \bool_lazy_or:nnT
6025     { \str_if_eq_p:ee { \l_tmpb_tl } { * } }
6026     { \tl_if_blank_p:o \l_tmpb_tl }
6027     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6028     \int_compare:nNnT { \l_tmpb_tl } > { #2 }
6029     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6030     \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
6031     { \clist_put_right:Nn #1 { #####1 } }
6032   }
6033 }

```

The following command will be linked to `\cellcolor` in the tabular.

```

6034 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
6035 {
6036   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6037   {

```

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).

```

6038     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
6039     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6040   }
6041   \ignorespaces
6042 }

6043 \NewDocumentCommand \@@_cellcolor_error { 0 { } m }
6044 { \@@_error:n { cellcolor~in~Block } }
6045 % \end{macrocode}
6046 %
6047 % \begin{macrocode}
6048 \NewDocumentCommand \@@_rowcolor_error { 0 { } m }
6049 { \@@_error:n { rowcolor~in~Block } }
6050 % \end{macrocode}
6051 %
6052 % \bigskip
6053 % The following command will be linked to |\rowcolor| in the tabular.
6054 % \begin{macrocode}
6055 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
6056 {
6057   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6058   {
6059     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
6060     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6061     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
6062   }
6063   \ignorespaces
6064 }

```

The following command will be linked to \rowcolors in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

6065 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
6066 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around #2 and #3 are mandatory.

The following command will be linked to \rowlistcolors in the tabular.

```

6067 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
6068 {

```

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.

```

6069   \seq_gclear:N \g_tmpa_seq
6070   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6071   { \@@_rowlistcolors_tabular:nnnn #1 }
6072   \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.

```

6073   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
6074   {
6075     { \int_use:N \c@iRow }
6076     { \exp_not:n { #1 } }
6077     { \exp_not:n { #2 } }
6078     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
6079   }

```

```

6080 \ignorespaces
6081 }

```

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`).

```

6082 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
6083 {
6084   \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).

```

6085   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
6086   {
6087     \tl_gput_right:Ne \g_@@_pre_code_before_tl
6088     {
6089       \@@_rowlistcolors
6090       [ \exp_not:n { #2 } ]
6091       { #1 - \int_eval:n { \c@iRow - 1 } }
6092       { \exp_not:n { #3 } }
6093       [ \exp_not:n { #4 } ]
6094     }
6095   }
6096 }

```

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.

```

6097 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
6098 {
6099   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6100   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
6101   \seq_gclear:N \g_@@_rowlistcolors_seq
6102 }

6103 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6104 {
6105   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6106   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6107 }

```

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.

```

6108 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
6109 {

```

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).

```

6110   \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
6111   {

```

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).

```

6112   \tl_gput_left:Ne \g_@@_pre_code_before_tl

```

```

6113     {
6114         \exp_not:N \columncolor [ #1 ]
6115         { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6116     }
6117 }
6118 }

6119 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6120 {
6121     \clist_map_inline:nn { #1 }
6122     {
6123         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6124         { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6125         \columncolor { nocolor } { ##1 }
6126     }
6127 }

6128 \cs_new_protected:Npn \@@_EmptyRow:n #1
6129 {
6130     \clist_map_inline:nn { #1 }
6131     {
6132         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6133         { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6134         \rowcolor { nocolor } { ##1 }
6135     }
6136 }

```

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`.

```

6137 \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`.

```

6138 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6139 {
6140     \int_if_zero:nTF { \l_@@_first_col_int }
6141     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6142     {
6143         \int_if_zero:nTF { \c@jCol }
6144         {
6145             \int_compare:nNnF { \c@iRow } = { -1 }
6146             {
6147                 \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int - 1 }
6148                 { #1 }
6149             }
6150         }
6151         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6152     }
6153 }

```

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.

```

6154 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6155 {
6156   \int_if_zero:nF { \c@iRow }
6157   {
6158     \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
6159     {
6160       \int_compare:nNnT { \c@jCol } > { \c_zero_int }
6161       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6162     }
6163   }
6164 }

```

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`.

```

6165 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6166 {
6167   \IfPackageLoadedTF { tikz }
6168   {
6169     \IfPackageLoadedTF { booktabs }
6170     { #2 }
6171     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6172   }
6173   { \@@_error:nn { TopRule~without~tikz } { #1 } }
6174 }
6175 \NewExpandableDocumentCommand { \@@_TopRule } { }
6176 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
6177 \cs_new:Npn \@@_TopRule_i:
6178 {
6179   \noalign \bgroup
6180   \peek_meaning:NTF [
6181   { \@@_TopRule_ii: }
6182   { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6183 }
6184 \NewDocumentCommand \@@_TopRule_ii: { o }
6185 {
6186   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6187   {
6188     \@@_hline:n
6189     {
6190       position = \int_eval:n { \c@iRow + 1 } ,
6191       tikz =
6192       {
6193         line-width = #1 ,
6194         yshift = 0.25 \arrayrulewidth ,
6195         shorten-< = - 0.5 \arrayrulewidth
6196       } ,
6197       total-width = #1
6198     }
6199   }
6200   \skip_vertical:n { \belowrulesep + #1 }
6201   \egroup
6202 }
6203 \NewExpandableDocumentCommand { \@@_BottomRule } { }
6204 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }

```

```

6205 \cs_new:Npn \@@_BottomRule_i:
6206 {
6207   \noalign \bgroup
6208   \peek_meaning:NTF [
6209     { \@@_BottomRule_ii: }
6210     { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6211   }
6212 \NewDocumentCommand \@@_BottomRule_ii: { o }
6213 {
6214   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6215   {
6216     \@@_hline:n
6217     {
6218       position = \int_eval:n { \c@iRow + 1 } ,
6219       tikz =
6220       {
6221         line-width = #1 ,
6222         yshift = 0.25 \arrayrulewidth ,
6223         shorten~< = - 0.5 \arrayrulewidth
6224       } ,
6225       total-width = #1 ,
6226     }
6227   }
6228   \skip_vertical:N \aboverulesep
6229   \@@_create_row_node_i:
6230   \skip_vertical:n { #1 }
6231   \egroup
6232 }
6233 \NewExpandableDocumentCommand { \@@_MidRule } { } { }
6234 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6235 \cs_new:Npn \@@_MidRule_i:
6236 {
6237   \noalign \bgroup
6238   \peek_meaning:NTF [
6239     { \@@_MidRule_ii: }
6240     { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6241   }
6242 \NewDocumentCommand \@@_MidRule_ii: { o }
6243 {
6244   \skip_vertical:N \aboverulesep
6245   \@@_create_row_node_i:
6246   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6247   {
6248     \@@_hline:n
6249     {
6250       position = \int_eval:n { \c@iRow + 1 } ,
6251       tikz =
6252       {
6253         line-width = #1 ,
6254         yshift = 0.25 \arrayrulewidth ,
6255         shorten~< = - 0.5 \arrayrulewidth
6256       } ,
6257       total-width = #1 ,
6258     }
6259   }
6260   \skip_vertical:n { \belowrulesep + #1 }
6261   \egroup
6262 }

```

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.

```

6263 \keys_define:nn { nicematrix / Rules }
6264 {
6265   position .int_set:N = \l_@@_position_int ,
6266   position .value_required:n = true ,
6267   start .int_set:N = \l_@@_start_int ,
6268   end .code:n =
6269     \bool_lazy_or:nnTF
6270     { \tl_if_empty_p:n { #1 } }
6271     { \str_if_eq_p:ee { #1 } { last } }
6272     { \int_set_eq:NN \l_@@_end_int \c@jCol }
6273     { \int_set:Nn \l_@@_end_int { #1 } }
6274 }

```

It’s possible that the rule won’t be drawn continuously from `start` to `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That’s why an analysis is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```

6275 \keys_define:nn { nicematrix / RulesBis }
6276 {
6277   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6278   multiplicity .initial:n = 1 ,
6279   dotted .bool_set:N = \l_@@_dotted_bool ,
6280   dotted .initial:n = false ,
6281   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]`.

```

6282   color .code:n =
6283     \@@_set_CTarc:n { #1 }
6284     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6285   color .value_required:n = true ,
6286   sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6287   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.

```

6288   tikz .code:n =
6289     \IfPackageLoadedTF { tikz }
6290     { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6291     { \@@_error:n { tikz~without~tikz } } ,
6292   tikz .value_required:n = true ,
6293   total-width .dim_set:N = \l_@@_rule_width_dim ,
6294   total-width .value_required:n = true ,
6295   width .meta:n = { total-width = #1 } ,
6296   unknown .code:n =
6297     \@@_unknown_key:nn
6298     { nicematrix / RulesBis }
6299     { Unknown-key-for-RulesBis }
6300 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```
6301 \cs_new_protected:Npn \@@_vline:n #1
6302 {
```

The group is for the options.

```
6303   \group_begin:
6304   \int_set_eq:NN \l_@@_end_int \c@iRow
6305   \keys_set_known:nN { 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).

```
6306   \int_compare:nNnT { \l_@@_position_int } < { \c@jCol + 2 }
6307   \@@_vline_i:
6308   \group_end:
6309 }

6310 \cs_new_protected:Npn \@@_vline_i:
6311 {
```

`\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`.

```
6312   \tl_set:N \l_tmpb_tl { \int_use:N \l_@@_position_int }
6313   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6314   \l_tmpa_tl
6315   {
```

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.

```
6316       \bool_gset_true:N \g_tmpa_bool
6317       \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6318       { \@@_test_vline_in_block:nnnnn ##1 }
6319       \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6320       { \@@_test_vline_in_block:nnnnn ##1 }
6321       \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6322       { \@@_test_vline_in_stroken_block:nnnn ##1 }
6323       \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_v: }
6324       \bool_if:NTF \g_tmpa_bool
6325       {
6326         \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.

```
6327         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6328       }
6329     {
6330       \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6331       {
6332         \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6333         \@@_vline_ii:
6334         \int_zero:N \l_@@_local_start_int
6335       }
6336     }
6337   }
6338   \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6339   {
6340     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6341     \@@_vline_ii:
6342   }
6343 }
```



```

6344 \cs_new_protected:Npn \@@_test_in_corner_v:
6345 {
6346   \int_compare:nNnTF { \l_tmpb_tl } = { \c@jCol + 1 }
6347   {
6348     \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6349     { \bool_set_false:N \g_tmpa_bool }
6350   }
6351   {
6352     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6353     {
6354       \int_compare:nNnTF { \l_tmpb_tl } = { \c_one_int }
6355       { \bool_set_false:N \g_tmpa_bool }
6356       {
6357         \@@_if_in_corner:nT
6358         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6359         { \bool_set_false:N \g_tmpa_bool }
6360       }
6361     }
6362   }
6363 }

6364 \cs_new_protected:Npn \@@_vline_ii:
6365 {
6366   \tl_clear:N \l_@@_tikz_rule_tl
6367   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6368   \bool_if:NTF \l_@@_dotted_bool
6369   { \@@_vline_iv: }
6370   {
6371     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6372     { \@@_vline_iii: }
6373     { \@@_vline_v: }
6374   }
6375 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

6376 \cs_new_protected:Npn \@@_vline_iii:
6377 {
6378   \pgfpicture
6379   \pgfrememberpicturepositiononpagetrue
6380   \pgf@relevantforpicturesizefalse
6381   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6382   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6383   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6384   \dim_set:Nn \l_tmpb_dim
6385   {
6386     \pgf@x
6387     - 0.5 \l_@@_rule_width_dim
6388     +
6389     ( \arrayrulewidth * \l_@@_multiplicity_int
6390       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6391   }
6392   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6393   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6394   \bool_lazy_all:nT
6395   {
6396     { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6397     { \cs_if_exist_p:N \CT@drsc@ }
6398     { ! \tl_if_blank_p:o \CT@drsc@ }
6399   }
6400   {
6401     \group_begin:
6402     \CT@drsc@

```

```

6403 \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6404 \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6405 \dim_set:Nn \l_@@_tmpd_dim
6406 {
6407   \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6408   * ( \l_@@_multiplicity_int - 1 )
6409 }
6410 \pgfpathrectanglecorners
6411 { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6412 { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6413 \pgfusepath { fill }
6414 \group_end:
6415 }
6416 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6417 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6418 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6419 {
6420   \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6421   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6422   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6423 }
6424 \CT@arc@
6425 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6426 \pgfsetrectcap
6427 \pgfusepathqstroke
6428 \endpgfpicture
6429 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6430 \cs_new_protected:Npn \@@_vline_iv:
6431 {
6432   \pgfpicture
6433   \pgfrememberpicturepositiononpagetrue
6434   \pgf@relevantforpicturesizefalse
6435   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6436   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6437   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6438   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6439   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6440   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6441   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6442   \CT@arc@
6443   \@@_draw_line:
6444   \endpgfpicture
6445 }

```

The following code is for the case when the user uses the key `tikz`.

```

6446 \cs_new_protected:Npn \@@_vline_v:
6447 {
6448   \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).

```

6449 \CT@arc@
6450 \tl_if_empty:NF \l_@@_rule_color_tl
6451 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6452 \pgfrememberpicturepositiononpagetrue
6453 \pgf@relevantforpicturesizefalse
6454 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6455 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6456 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6457 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }

```

```

6458 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6459 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6460 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6461 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6462 ( \l_tmpb_dim , \l_tmpa_dim ) --
6463 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6464 \end { tikzpicture }
6465 }

```

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).

```

6466 \cs_new_protected:Npn \@@_draw_vlines:
6467 {
6468   \int_step_inline:nnn
6469   {
6470     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6471     { 2 }
6472     { 1 }
6473   }
6474   {
6475     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6476     { \c@jCol }
6477     { \int_eval:n { \c@jCol + 1 } }
6478   }
6479   {
6480     \str_if_eq:eeF { \l_@@_vlines_clist } { all }
6481     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6482     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6483   }
6484 }

```

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}`.

```

6485 \cs_new_protected:Npn \@@_hline:n #1
6486 {

```

The group is for the options.

```

6487   \group_begin:
6488   \int_set_eq:NN \l_@@_end_int \c@jCol
6489   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6490   \@@_hline_i:
6491   \group_end:
6492 }

```

```

6493 \cs_new_protected:Npn \@@_hline_i:
6494 {

```

`\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`.

```

6495   \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6496   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6497   \l_tmpb_tl
6498   {

```

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.

```

6499   \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6500 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6501 { \@@_test_hline_in_block:nnnnn ##1 }

6502 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6503 { \@@_test_hline_in_block:nnnnn ##1 }
6504 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6505 { \@@_test_hline_in_stroken_block:nnnn ##1 }
6506 \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_h: }
6507 \bool_if:NTF \g_tmpa_bool
6508 {
6509 \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.

```

6510 { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6511 }
6512 {
6513 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6514 {
6515 \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6516 \@@_hline_ii:
6517 \int_zero:N \l_@@_local_start_int
6518 }
6519 }
6520 }
6521 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6522 {
6523 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6524 \@@_hline_ii:
6525 }
6526 }

```

```

6527 \cs_new_protected:Npn \@@_test_in_corner_h:
6528 {
6529 \int_compare:nNnTF { \l_tmpa_tl } = { \c@iRow + 1 }
6530 {
6531 \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6532 { \bool_set_false:N \g_tmpa_bool }
6533 }
6534 {
6535 \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6536 {
6537 \int_compare:nNnTF { \l_tmpa_tl } = { \c_one_int }
6538 { \bool_set_false:N \g_tmpa_bool }
6539 {
6540 \@@_if_in_corner:nT
6541 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6542 { \bool_set_false:N \g_tmpa_bool }
6543 }
6544 }
6545 }
6546 }

```

```

6547 \cs_new_protected:Npn \@@_hline_ii:
6548 {
6549 \tl_clear:N \l_@@_tikz_rule_tl
6550 \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6551 \bool_if:NTF \l_@@_dotted_bool
6552 { \@@_hline_iv: }
6553 {
6554 \tl_if_empty:NTF \l_@@_tikz_rule_tl

```

```

6555         { \@@_hline_iii: }
6556         { \@@_hline_v: }
6557     }
6558 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6559 \cs_new_protected:Npn \@@_hline_iii:
6560 {
6561     \pgfpicture
6562     \pgfrememberpicturepositiononpagetrue
6563     \pgf@relevantforpicturesizefalse
6564     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6565     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6566     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6567     \dim_set:Nn \l_tmpb_dim
6568     {
6569         \pgf@y
6570         - 0.5 \l_@@_rule_width_dim
6571         +
6572         ( \arrayrulewidth * \l_@@_multiplicity_int
6573           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6574     }
6575     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6576     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6577     \bool_lazy_all:nT
6578     {
6579         { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6580         { \cs_if_exist_p:N \CT@drsc@ }
6581         { ! \tl_if_blank_p:o \CT@drsc@ }
6582     }
6583     {
6584         \group_begin:
6585         \CT@drsc@
6586         \dim_set:Nn \l_@@_tmpd_dim
6587         {
6588             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6589             * ( \l_@@_multiplicity_int - 1 )
6590         }
6591         \pgfpathrectanglecorners
6592         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6593         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6594         \pgfusepathqfill
6595         \group_end:
6596     }
6597     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6598     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6599     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6600     {
6601         \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6602         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6603         \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6604     }
6605     \CT@arc@
6606     \pgfsetlinewidth { 1.1 \arrayrulewidth }
6607     \pgfsetrectcap
6608     \pgfusepathqstroke
6609     \endpgfpicture
6610 }

```

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6611 \cs_new_protected:Npn \@@_hline_iv:
6612 {
6613   \pgfpicture
6614   \pgfrememberpicturepositiononpagetrue
6615   \pgf@relevantforpicturesizefalse
6616   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6617   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6618   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6619   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6620   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6621   \int_compare:nNt { \l_@@_local_start_int } = { \c_one_int }
6622   {
6623     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6624     \bool_if:NF \g_@@_delims_bool
6625     { \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.

```

6626   \tl_if_eq:NnF \g_@@_left_delim_tl (
6627     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6628   )
6629   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6630   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6631   \int_compare:nNt { \l_@@_local_end_int } = { \c_jCol }
6632   {
6633     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6634     \bool_if:NF \g_@@_delims_bool
6635     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6636     \tl_if_eq:NnF \g_@@_right_delim_tl )
6637     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6638   }
6639   \CT@arc@
6640   \@@_draw_line:
6641   \endpgfpicture
6642 }

```

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`).

```

6643 \cs_new_protected:Npn \@@_hline_v:
6644 {
6645   \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).

```

6646   \CT@arc@
6647   \tl_if_empty:NF \l_@@_rule_color_tl

```

```

6648     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6649     \pgfrememberpicturepositiononpagetrue
6650     \pgf@relevantforpicturesizefalse
6651     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6652     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6653     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6654     \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6655     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6656     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6657     \exp_args:No \tikzset \l_@@_tikz_rule_tl
6658     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6659     ( \l_tmpa_dim , \l_tmpb_dim ) --
6660     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6661     \end { tikzpicture }
6662 }

```

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).

```

6663 \cs_new_protected:Npn \@@_draw_hlines:
6664 {
6665     \int_step_inline:nnn
6666     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6667     {
6668         \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6669         { \c@iRow }
6670         { \int_eval:n { \c@iRow + 1 } }
6671     }
6672     {
6673         \str_if_eq:eeF { \l_@@_hlines_clist } { all }
6674         { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6675         { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6676     }
6677 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6678 \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.

```

6679 \cs_set:Npn \@@_Hline_i:n #1
6680 {
6681     \peek_remove_spaces:n
6682     {
6683         \peek_meaning:NTF \Hline
6684         { \@@_Hline_ii:nn { #1 + 1 } }
6685         { \@@_Hline_iii:n { #1 } }
6686     }
6687 }
6688 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6689 \cs_set:Npn \@@_Hline_iii:n #1
6690 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6691 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6692 {
6693     \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6694     \skip_vertical:N \l_@@_rule_width_dim
6695     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6696     {
6697         \@@_hline:n
6698         {
6699             multiplicity = #1 ,
6700             position = \int_eval:n { \c@iRow + 1 } ,

```

```

6701         total-width = \dim_use:N \l_@@_rule_width_dim ,
6702         #2
6703     }
6704 }
6705 \egroup
6706 }

```

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`).

```

6707 \cs_new_protected:Npn \@@_custom_line:n #1
6708 {
6709     \str_clear_new:N \l_@@_command_str
6710     \str_clear_new:N \l_@@_ccommand_str
6711     \str_clear_new:N \l_@@_letter_str
6712     \tl_clear_new:N \l_@@_other_keys_tl
6713     \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
6714     \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
6715     {
6716         \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }

```

We delete the last character which is a space.

```

6717         \str_set:Ne \l_@@_command_str
6718         { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
6719     }
6720     \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
6721     {
6722         \str_set:Ne \l_@@_ccommand_str
6723         { \str_tail:N \l_@@_ccommand_str }
6724         \str_set:Ne \l_@@_ccommand_str
6725         { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
6726     }

```

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.

```

6727     \bool_lazy_all:nTF
6728     {
6729         { \str_if_empty_p:N \l_@@_letter_str }
6730         { \str_if_empty_p:N \l_@@_command_str }
6731         { \str_if_empty_p:N \l_@@_ccommand_str }
6732     }
6733     { \@@_error:n { No~letter~and~no~command } }
6734     { \@@_custom_line_i:o \l_@@_other_keys_tl }
6735 }
6736 \keys_define:nn { nicematrix / custom-line }
6737 {
6738     letter .str_set:N = \l_@@_letter_str ,
6739     letter .value_required:n = true ,
6740     command .str_set:N = \l_@@_command_str ,
6741     command .value_required:n = true ,
6742     ccommand .str_set:N = \l_@@_ccommand_str ,
6743     ccommand .value_required:n = true ,
6744 }

```

```

6745 \cs_new_protected:Npn \@@_custom_line_i:n #1
6746 {

```


The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6747 \bool_set_false:N \l_@@_tikz_rule_bool
6748 \bool_set_false:N \l_@@_dotted_rule_bool
6749 \bool_set_false:N \l_@@_color_bool
6750 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6751 \bool_if:NT \l_@@_tikz_rule_bool
6752 {
6753   \IfPackageLoadedF { tikz }
6754   { \@@_error:n { tikz~in~custom-line~without~tikz } }
6755   \bool_if:NT \l_@@_color_bool
6756   { \@@_error:n { color~in~custom-line~with~tikz } }
6757 }
6758 \bool_if:NT \l_@@_dotted_rule_bool
6759 {
6760   \int_compare:nNtT { \l_@@_multiplicity_int } > { \c_one_int }
6761   { \@@_error:n { key~multiplicity~with~dotted } }
6762 }
6763 \str_if_empty:NF \l_@@_letter_str
6764 {
6765   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6766   { \@@_error:n { Several~letters } }
6767   {
6768     \tl_if_in:NoTF
6769     \c_@@_forbidden_letters_str
6770     \l_@@_letter_str
6771     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6772   }

```

During the analysis of the preamble provided by the final user, our automaton, for the letter corresponding at the custom line, will directly use the following command that you define in the main hash table of TeX.

```

6773 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6774 { \@@_v_custom_line:nn { #1 } }
6775 }
6776 }
6777 }
6778 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6779 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6780 }
6781 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6782 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6783 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{nicematrix/Rules}`). That's why the following set of keys has some keys which are no-op.

```

6784 \keys_define:nn { nicematrix / custom-line-bis }
6785 {
6786   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6787   multiplicity .initial:n = 1 ,
6788   multiplicity .value_required:n = true ,
6789   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6790   color .value_required:n = true ,
6791   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6792   tikz .value_required:n = true ,
6793   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6794   dotted .value_forbidden:n = true ,
6795   total-width .code:n = { } ,
6796   total-width .value_required:n = true ,
6797   width .code:n = { } ,

```

```

6798 width .value_required:n = true ,
6799 sep-color .code:n = { } ,
6800 sep-color .value_required:n = true ,
6801 unknown .code:n =
6802   \@@_unknown_key:nn
6803   { nicematrix / custom-line-bis }
6804   { Unknown-key-for-custom-line }
6805 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6806 \bool_new:N \l_@@_dotted_rule_bool
6807 \bool_new:N \l_@@_tikz_rule_bool
6808 \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`.

```

6809 \keys_define:nn { nicematrix / custom-line-width }
6810 {
6811   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6812   multiplicity .initial:n = 1 ,
6813   multiplicity .value_required:n = true ,
6814   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6815   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6816                       \bool_set_true:N \l_@@_total_width_bool ,
6817   total-width .value_required:n = true ,
6818   width .meta:n = { total-width = #1 } ,
6819   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6820 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule (hence the ‘h’ in the name) with the full width of the array. `#1` is the whole set of keys to pass to the command `\@@_hline:n` (which is in the internal `\CodeAfter`).

```

6821 \cs_new_protected:Npn \@@_h_custom_line:n #1
6822 {

```

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`).

```

6823   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6824   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6825 }

```

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`).

```

6826 \cs_new_protected:Npn \@@_c_custom_line:n #1
6827 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

6828   \exp_args:Nc \NewExpandableDocumentCommand
6829   { nicematrix - \l_@@_ccommand_str }
6830   { 0 { } m }
6831   {
6832     \noalign
6833     {
6834       \@@_compute_rule_width:n { #1 , ##1 }
6835       \skip_vertical:n { \l_@@_rule_width_dim }
6836       \clist_map_inline:nn
6837       { ##2 }
6838       { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }

```

```

6839     }
6840   }
6841   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6842 }

```

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*.

```

6843 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6844 {
6845   \tl_if_in:nnTF { #2 } { - }
6846   { \@@_cut_on_hyphen:w #2 \q_stop }
6847   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6848   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6849   {
6850     \@@_hline:n
6851     {
6852       #1 ,
6853       start = \l_tmpa_tl ,
6854       end = \l_tmpb_tl ,
6855       position = \int_eval:n { \c@iRow + 1 } ,
6856       total-width = \dim_use:N \l_@@_rule_width_dim
6857     }
6858   }
6859 }

6860 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6861 {
6862   \bool_set_false:N \l_@@_tikz_rule_bool
6863   \bool_set_false:N \l_@@_total_width_bool
6864   \bool_set_false:N \l_@@_dotted_rule_bool
6865   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6866   \bool_if:NF \l_@@_total_width_bool
6867   {
6868     \bool_if:NTF \l_@@_dotted_rule_bool
6869     { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6870     {
6871       \bool_if:NF \l_@@_tikz_rule_bool
6872       {
6873         \dim_set:Nn \l_@@_rule_width_dim
6874         {
6875           \arrayrulewidth * \l_@@_multiplicity_int
6876           + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6877         }
6878       }
6879     }
6880   }
6881 }

```

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

```

6882 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
6883 {
6884   \str_if_eq:nnTF { #2 } { [ ] }
6885   { \@@_v_custom_line_i:nw { #1 } [ ] }
6886   { \@@_v_custom_line_ii:nn { #2 } { #1 } }
6887 }
6888 \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
6889 { \@@_v_custom_line:nn { #1 , #2 } }
6890 \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
6891 {
6892   \@@_compute_rule_width:n { #2 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6893   \tl_gput_right:Ne \g_@@_array_preamble_tl

```

```

6894     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6895 \tl_gput_right:Ne \g_@@_pre_code_after_tl
6896 {
6897     \@@_vline:n
6898     {
6899         #2 ,
6900         position = \int_eval:n { \c@jCol + 1 } ,
6901         total-width = \dim_use:N \l_@@_rule_width_dim
6902     }
6903 }
6904 \@@_rec_preamble:n #1
6905 }
6906 \@@_custom_line:n
6907 { 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.

```

6908 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6909 {
6910     \int_compare:nNnT { \l_tmpa_tl } > { #1 }
6911     {
6912         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6913         {
6914             \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6915             {
6916                 \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6917                 { \bool_gset_false:N \g_tmpa_bool }
6918             }
6919         }
6920     }
6921 }

```

The same for vertical rules.

```

6922 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6923 {
6924     \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6925     {
6926         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6927         {
6928             \int_compare:nNnT { \l_tmpb_tl } > { #2 }
6929             {
6930                 \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6931                 { \bool_gset_false:N \g_tmpa_bool }
6932             }
6933         }
6934     }
6935 }
6936 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6937 {
6938     \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6939     {
6940         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6941         {
6942             \int_compare:nNnTF { \l_tmpa_tl } = { #1 }
6943             { \bool_gset_false:N \g_tmpa_bool }
6944             {
6945                 \int_compare:nNnT { \l_tmpa_tl } = { #3 + 1 }
6946                 { \bool_gset_false:N \g_tmpa_bool }
6947             }
6948         }
6949     }
6950 }

```

```

6948     }
6949   }
6950 }
6951 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6952 {
6953   \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6954   {
6955     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6956     {
6957       \int_compare:nNnTF { \l_tmpb_tl } = { #2 }
6958       { \bool_gset_false:N \g_tmpa_bool }
6959       {
6960         \int_compare:nNnT { \l_tmpb_tl } = { #4 + 1 }
6961         { \bool_gset_false:N \g_tmpa_bool }
6962       }
6963     }
6964   }
6965 }

```

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.

```

6966 \cs_new_protected:Npn \@@_compute_corners:
6967 {
6968   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6969   { \@@_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`).

```

6970   \clist_clear:N \l_@@_corners_cells_clist
6971   \clist_map_inline:Nn \l_@@_corners_clist
6972   {
6973     \str_case:nnF { ##1 }
6974     {
6975       { NW }
6976       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6977       { NE }
6978       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6979       { SW }
6980       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6981       { SE }
6982       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6983     }
6984     { \@@_error:nn { bad~corner } { ##1 } }
6985   }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6986   \clist_if_empty:NF \l_@@_corners_cells_clist
6987   {

```

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.

```

6988     \tl_gput_right:Ne \g_@@_aux_tl
6989     {
6990       \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist

```

```

6991         { \l_@@_corners_cells_clist }
6992     }
6993 }
6994 }

6995 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6996 {
6997     \int_step_inline:nnn { #1 } { #3 }
6998     {
6999         \int_step_inline:nnn { #2 } { #4 }
7000         { \cs_set_nopar:cpn { @@ _ block _ ##1 - ###1 } { } }
7001     }
7002 }

7003 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
7004 {
7005     \cs_if_exist:cTF
7006     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
7007     { \prg_return_true: }
7008     { \prg_return_false: }
7009 }

```

“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.

```

7010 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
7011 {

```

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.

```

7012     \bool_set_false:N \l_tmpa_bool
7013     \int_zero_new:N \l_@@_last_empty_row_int
7014     \int_set:Nn \l_@@_last_empty_row_int { #1 }
7015     \int_step_inline:nnnn { #1 } { #3 } { #5 }
7016     {
7017         \bool_lazy_or:nnTF
7018         {
7019             \cs_if_exist_p:c
7020             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
7021         }
7022         { \@@_if_in_block_p:nn { ##1 } { #2 } }
7023         { \bool_set_true:N \l_tmpa_bool }
7024         {
7025             \bool_if:NF \l_tmpa_bool
7026             { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
7027         }
7028     }

```

Now, you determine the last empty cell in the row of number 1.

```

7029 \bool_set_false:N \l_tmpa_bool
7030 \int_zero_new:N \l_@@_last_empty_column_int
7031 \int_set:Nn \l_@@_last_empty_column_int { #2 }
7032 \int_step_inline:nnnn { #2 } { #4 } { #6 }
7033 {
7034   \bool_lazy_or:nnTF
7035   {
7036     \cs_if_exist_p:c
7037     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
7038   }
7039   { \@@_if_in_block_p:nn { #1 } { ##1 } }
7040   { \bool_set_true:N \l_tmpa_bool }
7041   {
7042     \bool_if:NF \l_tmpa_bool
7043     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
7044   }
7045 }

```

Now, we loop over the rows.

```

7046 \int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int }
7047 {

```

We treat the row number ##1 with another loop.

```

7048 \bool_set_false:N \l_tmpa_bool
7049 \int_step_inline:nnnn { #2 } { #4 } { \l_@@_last_empty_column_int }
7050 {
7051   \bool_lazy_or:nnTF
7052   { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7053   { \@@_if_in_block_p:nn { ##1 } { #####1 } }
7054   { \bool_set_true:N \l_tmpa_bool }
7055   {
7056     \bool_if:NF \l_tmpa_bool
7057     {
7058       \int_set:Nn \l_@@_last_empty_column_int { #####1 }
7059       \clist_put_right:Nn
7060       \l_@@_corners_cells_clist
7061       { ##1 - #####1 }
7062       \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7063     }
7064   }
7065 }
7066 }
7067 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

7068 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
7069 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }

```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient: `\clist_if_in:NnT \l_@@_corners_cells_clist { #1 } ...`

24 The environment {NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```

7070 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

7071 \keys_define:nn { nicematrix / NiceMatrixBlock }
7072 {
7073   auto-columns-width .code:n =
7074   {
7075     \bool_set_true:N \l_@@_block_auto_columns_width_bool
7076     \dim_gzero_new:N \g_@@_max_cell_width_dim
7077     \bool_set_true:N \l_@@_auto_columns_width_bool
7078   }
7079 }

7080 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
7081 {
7082   \int_gincr:N \g_@@_NiceMatrixBlock_int
7083   \dim_zero:N \l_@@_columns_width_dim
7084   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
7085   \bool_if:NT \l_@@_block_auto_columns_width_bool
7086   {
7087     \cs_if_exist:cT
7088     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
7089     {
7090       \dim_set:Nn \l_@@_columns_width_dim
7091       {
7092         \use:c
7093         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7094       }
7095     }
7096   }
7097 }

```

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`).

```

7098 {
7099   \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.

```

7100   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
7101   {
7102     \bool_if:NT \l_@@_block_auto_columns_width_bool
7103     {
7104       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
7105       \iow_shipout:Ne \@mainaux
7106       {
7107         \cs_gset:cpn
7108         { @@ _ 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.

```

7109         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7110       }
7111       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
7112     }
7113   }
7114   \ignorespacesafterend
7115 }

```


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).

```

7116 \cs_new_protected:Npn \@@_create_extra_nodes:
7117 {
7118   \bool_if:nTF \l_@@_medium_nodes_bool
7119   {
7120     \bool_if:NTF \l_@@_no_cell_nodes_bool
7121     { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7122     {
7123       \bool_if:NTF \l_@@_large_nodes_bool
7124       \@@_create_medium_and_large_nodes:
7125       \@@_create_medium_nodes:
7126     }
7127   }
7128   {
7129     \bool_if:NT \l_@@_large_nodes_bool
7130     {
7131       \bool_if:NTF \l_@@_no_cell_nodes_bool
7132       { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7133       \@@_create_large_nodes:
7134     }
7135   }
7136 }

```

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`.

```

7137 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7138 {
7139   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7140   {
7141     \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7142     \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7143     \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7144     \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7145   }
7146   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7147   {
7148     \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7149     \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7150     \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7151     \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7152   }

```

We begin the two nested loops over the rows and the columns of the array.

```

7153   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7154   {
7155       \int_step_variable:nnNn
7156       \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.

```

7157       {
7158           \cs_if_exist:cT
7159           { 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`.

```

7160       {
7161           \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7162           \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7163           { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7164           \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7165           {
7166               \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7167               { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7168           }

```

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`.

```

7169           \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7170           \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7171           { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } { \pgf@y } }
7172           \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7173           {
7174               \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7175               { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } { \pgf@x } }
7176           }
7177       }
7178   }
7179 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7180   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7181   {
7182       \dim_compare:nNnT
7183       { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7184       {
7185           \@@_qpoint:n { row - \@@_i: - base }
7186           \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7187           \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7188       }
7189   }
7190   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7191   {
7192       \dim_compare:nNnT
7193       { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7194       {
7195           \@@_qpoint:n { col - \@@_j: }
7196           \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7197           \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7198       }
7199   }
7200 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7201 \cs_new_protected:Npn \@@_create_medium_nodes:
7202 {
7203   \pgfpicture
7204   \pgfrememberpicturepositiononpagetrue
7205   \pgf@relevantforpicturesizefalse
7206   \@@_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”.

```

7207   \tl_set:Nn \l_@@_suffix_tl { -medium }
7208   \@@_create_nodes:
7209   \endpgfpicture
7210 }

```

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:`.

```

7211 \cs_new_protected:Npn \@@_create_large_nodes:
7212 {
7213   \pgfpicture
7214   \pgfrememberpicturepositiononpagetrue
7215   \pgf@relevantforpicturesizefalse
7216   \@@_computations_for_medium_nodes:
7217   \@@_computations_for_large_nodes:
7218   \tl_set:Nn \l_@@_suffix_tl { - large }
7219   \@@_create_nodes:
7220   \endpgfpicture
7221 }

7222 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7223 {
7224   \pgfpicture
7225   \pgfrememberpicturepositiononpagetrue
7226   \pgf@relevantforpicturesizefalse
7227   \@@_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”.

```

7228   \tl_set:Nn \l_@@_suffix_tl { - medium }
7229   \@@_create_nodes:
7230   \@@_computations_for_large_nodes:
7231   \tl_set:Nn \l_@@_suffix_tl { - large }
7232   \@@_create_nodes:
7233   \endpgfpicture
7234 }

```

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.

```

7235 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7236 {
7237   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7238   \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`.

```

7239   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7240   {
7241     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }

```

¹⁵If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7242     {
7243     (
7244         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7245         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7246     )
7247     / 2
7248     }
7249     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7250     { l_@@_row _ \@@_i: _ min_dim }
7251 }
7252 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7253 {
7254     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7255     {
7256     (
7257         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7258         \dim_use:c
7259         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7260     )
7261     / 2
7262     }
7263     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7264     { l_@@_column _ \@@_j: _ max _ dim }
7265 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7266     \dim_sub:cn
7267     { l_@@_column _ 1 _ min _ dim }
7268     \l_@@_left_margin_dim
7269     \dim_add:cn
7270     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7271     \l_@@_right_margin_dim
7272 }

```

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).

```

7273 \cs_new_protected:Npn \@@_create_nodes:
7274 {
7275     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7276     {
7277         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7278         {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7279         \@@_pgf_rect_node:nnnnn
7280         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7281         { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7282         { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7283         { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7284         { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7285         \str_if_empty:NF \l_@@_name_str
7286         {
7287             \pgfnodealias
7288             { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7289             { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7290         }
7291     }
7292 }
7293 \int_step_inline:nn { \c@iRow }

```

```

7294     {
7295         \pgfnodealias
7296         { \@@_env: - ##1 - last \l_@@_suffix_tl }
7297         { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7298     }
7299     \int_step_inline:nn { \c@jCol }
7300     {
7301         \pgfnodealias
7302         { \@@_env: - last - ##1 \l_@@_suffix_tl }
7303         { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7304     }
7305     \pgfnodealias % added 2025-04-05
7306     { \@@_env: - last - last \l_@@_suffix_tl }
7307     { \@@_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 .

```

7308     \seq_map_pairwise_function:NNN
7309     \g_@@_multicolumn_cells_seq
7310     \g_@@_multicolumn_sizes_seq
7311     \@@_node_for_multicolumn:nn
7312 }

7313 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7314 {
7315     \cs_set_nopar:Npn \@@_i: { #1 }
7316     \cs_set_nopar:Npn \@@_j: { #2 }
7317 }

```

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”).

```

7318 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7319 {
7320     \@@_extract_coords_values: #1 \q_stop
7321     \@@_pgf_rect_node:nnnnn
7322     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7323     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7324     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7325     { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7326     { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7327     \str_if_empty:NF \l_@@_name_str
7328     {
7329         \pgfnodealias
7330         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7331         { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7332     }
7333 }

```

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).

```

7334 \keys_define:nn { nicematrix / Block / FirstPass }
7335 {
7336   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7337     \bool_set_true:N \l_@@_p_block_bool ,
7338   j .value_forbidden:n = true ,
7339   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7340   l .value_forbidden:n = true ,
7341   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7342   r .value_forbidden:n = true ,
7343   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7344   c .value_forbidden:n = true ,
7345   L .code:n = \str_set:Nn \l_@@_hpos_block_str L ,
7346   L .value_forbidden:n = true ,
7347   R .code:n = \str_set:Nn \l_@@_hpos_block_str R ,
7348   R .value_forbidden:n = true ,
7349   C .code:n = \str_set:Nn \l_@@_hpos_block_str C ,
7350   C .value_forbidden:n = true ,
7351   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7352   t .value_forbidden:n = true ,
7353   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7354   T .value_forbidden:n = true ,
7355   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7356   b .value_forbidden:n = true ,
7357   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7358   B .value_forbidden:n = true ,
7359   m .code:n = \str_set:Nn \l_@@_vpos_block_str m ,
7360   m .value_forbidden:n = true ,
7361   v-center .meta:n = m ,
7362   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7363   p .value_forbidden:n = true ,
7364   color .code:n =
7365     \@@_color:n { #1 }
7366     \tl_set_rescan:Nnn
7367       \l_@@_draw_tl
7368       { \char_set_catcode_other:N ! }
7369       { #1 } ,
7370   color .value_required:n = true ,
7371   respect-arraystretch .code:n =
7372     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7373   respect-arraystretch .value_forbidden:n = true ,
7374 }

```

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.

```

7375 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7376 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7377 {

```

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).

```

7378   \tl_if_blank:nTF { #2 }
7379   { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7380   {
7381     \tl_if_in:nnTF { #2 } { - }
7382     {
7383       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7384       \@@_Block_i_czech:w \@@_Block_i:w
7385       #2 \q_stop
7386     }
7387     {
7388       \@@_error:nn { Bad~argument~for~Block } { #2 }

```

```

7389         \@@_Block_ii:nnnnn \c_one_int \c_one_int
7390     }
7391 }
7392 { #1 } { #3 } { #4 }
7393 \ignorespaces
7394 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7395 \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`.

```

7396 {
7397   \char_set_catcode_active:N -
7398   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7399 }

```

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.

```

7400 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7401 {

```

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).

```

7402   \bool_lazy_or:nnTF
7403     { \tl_if_blank_p:n { #1 } }
7404     { \str_if_eq_p:ee { * } { #1 } }
7405     { \int_set:Nn \l_tmpa_int { 100 } }
7406     { \int_set:Nn \l_tmpa_int { #1 } }
7407   \bool_lazy_or:nnTF
7408     { \tl_if_blank_p:n { #2 } }
7409     { \str_if_eq_p:ee { * } { #2 } }
7410     { \int_set:Nn \l_tmpb_int { 100 } }
7411     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7412   \int_compare:nNnTF { \l_tmpb_int } = { \c_one_int }
7413   {
7414     \tl_if_empty:NTF \l_@@_hpos_cell_tl
7415       { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7416       { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7417   }
7418   { \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.

```

7419   \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7420   \tl_set:Ne \l_tmpa_tl
7421   {
7422     { \int_use:N \c@iRow }
7423     { \int_use:N \c@jCol }
7424     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7425     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7426   }

```

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).

```

7427     \bool_set_false:N \l_tmpa_bool
7428     \bool_if:NT \l_@@_amp_in_blocks_bool
\tl_if_in:nnT is slightly faster than \str_if_in:nnT.
7429     { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7430     \bool_case:nF
7431     {
7432         \l_tmpa_bool                                { \@@_Block_vii:eennn }
7433         \l_@@_p_block_bool                            { \@@_Block_vi:eennn }

```

For the blocks mono-column, we will compose right away in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a `X` column, we should not do that since the width is determined by another way. This should be the same for the `p`, `m` and `b` columns and we should modify that point. However, for the `X` column, it’s imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

7434         \l_@@_X_bool                                { \@@_Block_v:eennn }
7435         { \tl_if_empty_p:n { #5 } }                  { \@@_Block_v:eennn }
7436         { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7437         { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7438     }
7439     { \@@_Block_v:eennn }
7440     { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7441 }

```

The following macro is for the case of a `\Block` which is mono-row or mono-column (or both) and don’t use the key `p`. In that case, the content of the block is composed right away in a box (because we have to take into account the dimensions of that box for the width of the current column or the height and the depth of the current row). However, that box will be put in the array *after the construction of the array* (by using PGF) with `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn` which will do the main job.

#1 is *i* (the number of rows of the block), **#2** is *j* (the number of columns of the block), **#3** is the list of `key=values` pairs, **#4** are the tokens to put before the potential math mode and before the composition of the block and **#5** is the label (=content) of the block.

```

7442 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7443 {
7444     \int_gincr:N \g_@@_block_box_int
7445     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
7446     \cs_set_eq:NN \rowcolor \@@_rowcolor_error
7447     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7448     {
7449         \tl_gput_right:Ne \g_@@_pre_code_after_tl
7450         {
7451             \@@_actually_diagbox:nnnnnn
7452             { \int_use:N \c@iRow }
7453             { \int_use:N \c@jCol }
7454             { \int_eval:n { \c@iRow + #1 - 1 } }
7455             { \int_eval:n { \c@jCol + #2 - 1 } }
7456             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7457             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7458         }
7459     }
7460     \box_gclear_new:c
7461     { g_@@_block_box_int _int_use:N \g_@@_block_box_int _box }

```


Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful*: if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7462   \hbox_gset:cn
7463   { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7464   {

```

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`).

```

7465   \tl_if_empty:NTF \l_@@_color_tl
7466   { \int_compare:nNnT { #2 } = { \c_one_int } { \set@color } }
7467   { \@@_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`.

```

7468   \int_compare:nNnT { #1 } = { \c_one_int }
7469   {
7470     \int_if_zero:nTF { \c@iRow }
7471     {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7472   \cs_set_eq:NN \Block \@@_NullBlock:
7473   \l_@@_code_for_first_row_tl
7474   }
7475   {
7476     \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7477     {
7478       \cs_set_eq:NN \Block \@@_NullBlock:
7479       \l_@@_code_for_last_row_tl
7480     }
7481   }
7482   \g_@@_row_style_tl
7483   }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7484   \@@_reset_arraystretch:
7485   \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7486   #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.).

```
7487 \@@_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}`.

```
7488 \bool_if:NTF \l_@@_tabular_bool
7489 {
7490   \bool_lazy_all:nTF
7491   {
7492     { \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.

```
7493   {
7494     ! \dim_compare_p:nNn
7495     { \l_@@_col_width_dim } < { \c_zero_dim }
7496   }
7497   { ! \g_@@_rotate_bool }
7498 }
```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```
7499 {
7500   \use:e
7501   {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
7502   \exp_not:N \begin { minipage }
7503   [ \str_lowercase:f \l_@@_vpos_block_str ]
7504   { \l_@@_col_width_dim }
7505   \str_case:on \l_@@_hpos_block_str
7506   { c \centering r \raggedleft l \raggedright }
7507   }
7508   #5
7509   \end { minipage }
7510 }
```

In the other cases, we use a `{tabular}`.

```
7511 {
7512   \use:e
7513   {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
7514   \exp_not:N \begin { tabular }
7515   [ \str_lowercase:f \l_@@_vpos_block_str ]
7516   { @ { } \l_@@_hpos_block_str @ { } }
7517   }
7518   #5
7519   \end { tabular }
7520 }
7521 }
```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```
7522 {
7523   $ % $
7524   \use:e
7525   {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7526         \exp_not:N \begin { array }
7527         [ \str_lowercase:f \l_@@_vpos_block_str ]
7528         { @ { } \l_@@_hpos_block_str @ { } }
7529     }
7530     #5
7531     \end { array }
7532     $ % $
7533 }
7534 }
```

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).

```

7535     \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.

```

7536     \int_compare:nNnT { #2 } = { \c_one_int }
7537     {
7538         \dim_gset:Nn \g_@@_blocks_wd_dim
7539         {
7540             \dim_max:nn
7541             { \g_@@_blocks_wd_dim }
7542             {
7543                 \box_wd:c
7544                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7545             }
7546         }
7547     }
```

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.

```

7548     \int_compare:nNnT { #1 } = { \c_one_int }
7549     {
7550         \bool_lazy_any:nT
7551         {
7552             { \str_if_empty_p:N \l_@@_vpos_block_str }
7553             { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
7554             { \str_if_eq_p:ee { \l_@@_vpos_block_str } { b } }
7555         }
7556         { \@@_adjust_blocks_ht_dp: }
7557     }
7558     \seq_gput_right:Ne \g_@@_blocks_seq
7559     {
7560         \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.

```

7561     {
7562         \exp_not:n { #3 } ,
7563         \l_@@_hpos_block_str ,
```

Now, we put a key for the vertical alignment.

```

7564     \bool_if:NT \g_@@_rotate_bool
7565     {
7566         \bool_if:NTF \g_@@_rotate_c_bool
7567         { m }
7568         {
```

```

7569         \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7570         { T }
7571     }
7572 }
7573 }
7574 {
7575     \box_use_drop:c
7576     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7577 }
7578 }
7579 \bool_set_false:N \g_@@_rotate_c_bool
7580 }

7581 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7582 {
7583     \dim_gset:Nn \g_@@_blocks_ht_dim
7584     {
7585         \dim_max:nn
7586         { \g_@@_blocks_ht_dim }
7587         {
7588             \box_ht:c
7589             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7590         }
7591     }
7592     \dim_gset:Nn \g_@@_blocks_dp_dim
7593     {
7594         \dim_max:nn
7595         { \g_@@_blocks_dp_dim }
7596         {
7597             \box_dp:c
7598             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7599         }
7600     }
7601 }

7602 \cs_new:Npn \@@_adjust_hpos_rotate:
7603 {
7604     \bool_if:NT \g_@@_rotate_bool
7605     {
7606         \str_set:Ne \l_@@_hpos_block_str
7607         {
7608             \bool_if:NTF \g_@@_rotate_c_bool
7609             { c }
7610             {
7611                 \str_case:onF \l_@@_vpos_block_str
7612                 { b l B l t r T r }
7613                 {
7614                     \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
7615                     { r }
7616                     { l }
7617                 }
7618             }
7619         }
7620     }
7621 }
7622 \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*.

```

7623 \cs_new_protected:Npn \@@_rotate_box_of_block:
7624 {
7625     \box_grotate:cn

```

```

7626 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7627 { 90 }
7628 \int_compare:nNt { \c@iRow } = { \l_@@_last_row_int }
7629 {
7630   \vbox_gset_top:cn
7631   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7632   {
7633     \skip_vertical:n { 0.8 ex }
7634     \box_use:c
7635     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7636   }
7637 }
7638 \bool_if:NT \g_@@_rotate_c_bool
7639 {
7640   \hbox_gset:cn
7641   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7642   {
7643     $ % $
7644     \vcenter
7645     {
7646       \box_use:c
7647       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7648     }
7649     $ % $
7650   }
7651 }
7652 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key p). In that case, the content of the block is *not* composed right away in a box. The composition in a box will be done further, just after the construction of the array (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnn`).

#1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7653 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7654 {
7655   \seq_gput_right:Ne \g_@@_blocks_seq
7656   {
7657     \l_tmpa_tl
7658     { \exp_not:n { #3 } }
7659     {
7660       \bool_if:NTF \l_@@_tabular_bool
7661       {
7662         \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7663 \@@_reset_arraystretch:
7664 \exp_not:n
7665 {
7666   \dim_zero:N \extrarowheight
7667   #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).

```

7668 \tag_if_active:T { \tag_stop:n { table } }
7669 \use:e
7670 {
7671   \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7672   { @ { } \l_@@_hpos_block_str @ { } }

```

```

7673         }
7674         #5
7675     \end { tabular }
7676 }
7677 \group_end:
7678 }

```

When we are *not* in an environment {NiceTabular} (or similar).

```

7679     {
7680     \group_begin:

```

The following will be no-op when respect-arraystretch is in force.

```

7681     \@@_reset_arraystretch:
7682     \exp_not:n
7683     {
7684         \dim_zero:N \extrarowheight
7685         #4
7686         $ % $
7687         \use:e
7688         {
7689             \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7690             { @ { } \l_@@_hpos_block_str @ { } }
7691         }
7692         #5
7693         \end { array }
7694         $ % $
7695     }
7696     \group_end:
7697 }
7698 }
7699 }
7700 }
7701 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a \Block which uses the key p.

```

7702 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7703 {
7704     \seq_gput_right:Ne \g_@@_blocks_seq
7705     {
7706         \l_tmpa_tl
7707         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7708         { { \exp_not:n { #4 #5 } } }
7709     }
7710 }
7711 \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.

```

7712 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7713 {
7714     \seq_gput_right:Ne \g_@@_blocks_seq
7715     {
7716         \l_tmpa_tl
7717         { \exp_not:n { #3 } }
7718         { \exp_not:n { #4 #5 } }
7719     }
7720 }
7721 \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).

```

7722 \keys_define:n { nicematrix / Block / SecondPass }
7723 {
7724     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7725     ampersand-in-blocks .default:n = true ,
7726     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7727     tikz .code:n =
7728         \IfPackageLoadedTF { tikz }
7729         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7730         { \@@_error:n { tikz~key~without~tikz } } ,
7731     tikz .value_required:n = true ,
7732     fill .code:n =
7733         \tl_set_rescan:Nnn
7734         \l_@@_fill_tl
7735         { \char_set_catcode_other:N ! }
7736         { #1 } ,
7737     fill .value_required:n = true ,

```

In fine, the opacity will be applied by `\pgfsetfillopacity`.

```

7738     opacity .tl_set:N = \l_@@_opacity_tl ,
7739     opacity .value_required:n = true ,
7740     draw .code:n =
7741         \tl_set_rescan:Nnn
7742         \l_@@_draw_tl
7743         { \char_set_catcode_other:N ! }
7744         { #1 } ,
7745     draw .default:n = default ,
7746     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7747     rounded-corners .default:n = 4 pt ,
7748     color .code:n =
7749         \@@_color:n { #1 }
7750         \tl_set_rescan:Nnn
7751         \l_@@_draw_tl
7752         { \char_set_catcode_other:N ! }
7753         { #1 } ,
7754     borders .clist_set:N = \l_@@_borders_clist ,
7755     borders .value_required:n = true ,
7756     hvlines .meta:n = { vlines , hlines } ,
7757     vlines .bool_set:N = \l_@@_vlines_block_bool ,
7758     vlines .default:n = true ,
7759     hlines .bool_set:N = \l_@@_hlines_block_bool ,
7760     hlines .default:n = true ,
7761     line-width .dim_set:N = \l_@@_line_width_dim ,
7762     line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7763     j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7764         \bool_set_true:N \l_@@_p_block_bool ,
7765     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7766     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7767     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7768     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7769         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7770     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7771         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7772     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7773         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7774     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7775     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7776     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7777     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7778     m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7779     m .value_forbidden:n = true ,
7780     v-center .meta:n = m ,

```

```

7781 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7782 p .value_forbidden:n = true ,
7783 name .tl_set:N = \l_@@_block_name_str , % .str_set:N ?
7784 name .value_required:n = true ,
7785 name .initial:n = ,
7786 respect-arraystretch .code:n =
7787   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7788 respect-arraystretch .value_forbidden:n = true ,
7789 transparent .bool_set:N = \l_@@_transparent_bool ,
7790 transparent .default:n = true ,
7791 transparent .initial:n = false ,
7792 unknown .code:n =
7793   \@@_unknown_key:nn
7794     { nicematrix / Block / SecondPass }
7795     { Unknown-key-for-Block }
7796 }

```

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.

```

7797 \cs_new_protected:Npn \@@_draw_blocks:
7798 {
7799   \bool_if:NTF \c_@@_revtex_bool
7800     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7801     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7802   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7803 }
7804 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7805 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7806   \int_zero:N \l_@@_last_row_int
7807   \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”).

```

7808   \int_compare:nNnTF { #3 } > { 98 }
7809     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7810     { \int_set:Nn \l_@@_last_row_int { #3 } }
7811   \int_compare:nNnTF { #4 } > { 98 }
7812     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7813     { \int_set:Nn \l_@@_last_col_int { #4 } }
7814   \int_compare:nNnTF { \l_@@_last_col_int } > { \g_@@_col_total_int }
7815     {
7816       \bool_lazy_and:nnTF
7817         { \l_@@_preamble_bool }
7818         {
7819           \int_compare_p:n
7820             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7821         }
7822     {
7823       \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7824       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7825       \@@_msg_redirect_name:nn { columns-not-used } { none }
7826     }
7827     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7828 }

```



```

7829 {
7830   \int_compare:nNnTF { \l_@@_last_row_int } > { \g_@@_row_total_int }
7831   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7832   {
7833     \@@_Block_v:nneenn
7834     { #1 }
7835     { #2 }
7836     { \int_use:N \l_@@_last_row_int }
7837     { \int_use:N \l_@@_last_col_int }
7838     { #5 }
7839     { #6 }
7840   }
7841 }
7842 }

```

The following command `\@@_Block_v:nnnnnn` will actually draw the block. #1 is the first row of the block; #2 is the first column of the block; #3 is the last row of the block; #4 is the last column of the block; #5 is a list of `key=value` options; #6 is the label (content) of the block.

```

7843 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7844 {

```

The group is for the keys.

```

7845   \group_begin:
7846   \int_compare:nNnT { #1 } = { #3 }
7847   { \str_set:Nn \l_@@_vpos_block_str { t } }
7848   \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`.

```

7849   \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7850   \bool_lazy_and:nnT
7851   { \l_@@_vlines_block_bool }
7852   { ! \l_@@_ampersand_bool }
7853   {
7854     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7855     {
7856       \@@_vlines_block:nnn
7857       { \exp_not:n { #5 } }
7858       { #1 - #2 }
7859       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7860     }
7861   }
7862   \bool_if:NT \l_@@_hlines_block_bool
7863   {
7864     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7865     {
7866       \@@_hlines_block:nnn
7867       { \exp_not:n { #5 } }
7868       { #1 - #2 }
7869       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7870     }
7871   }
7872   \bool_if:NF \l_@@_transparent_bool
7873   {
7874     \bool_lazy_and:nnF { \l_@@_vlines_block_bool } { \l_@@_hlines_block_bool }
7875     {

```

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).

```

7876     \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7877     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7878   }
7879 }

```

```

7880 \tl_if_empty:NF \l_@@_draw_tl
7881 {
7882   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7883   { \@@_error:n { hlines~with~color } }
7884   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7885   {
7886     \@@_stroke_block:nnn
7887     { \exp_not:n { #5 } }
7888     { #1 - #2 }
7889     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7890   }
7891   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7892   { { #1 } { #2 } { #3 } { #4 } }
7893 }
7894 \clist_if_empty:NF \l_@@_borders_clist
7895 {
7896   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7897   {
7898     \@@_stroke_borders_block:nnn
7899     { \exp_not:n { #5 } }
7900     { #1 - #2 }
7901     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7902   }
7903 }
7904 \tl_if_empty:NF \l_@@_fill_tl
7905 {
7906   \@@_add_opacity_to_fill:
7907   \tl_gput_right:Ne \g_@@_pre_code_before_tl
7908   {
7909     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7910     { #1 - #2 }
7911     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7912     { \dim_use:N \l_@@_rounded_corners_dim }
7913   }
7914 }
7915 \seq_if_empty:NF \l_@@_tikz_seq
7916 {
7917   \tl_gput_right:Ne \g_nicematrix_code_before_tl
7918   {
7919     \@@_block_tikz:nnnnn
7920     { \seq_use:Nn \l_@@_tikz_seq { , } }
7921     { #1 }
7922     { #2 }
7923     { \int_use:N \l_@@_last_row_int }
7924     { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7925   }
7926 }
7927 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7928 {
7929   \tl_gput_right:Ne \g_@@_pre_code_after_tl
7930   {
7931     \@@_actually_diagbox:nnnnnn
7932     { #1 }
7933     { #2 }
7934     { \int_use:N \l_@@_last_row_int }
7935     { \int_use:N \l_@@_last_col_int }
7936     { \exp_not:n { ##1 } }
7937     { \exp_not:n { ##2 } }

```

```

7938     }
7939 }

```

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.

```

7940 \pgfpicture
7941 \pgfrememberpicturepositiononpagetrue
7942 \pgf@relevantforpicturesizefalse
7943 \@@_qpoint:n { row - #1 }
7944 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7945 \@@_qpoint:n { col - #2 }
7946 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7947 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7948 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7949 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7950 \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.

```

7951 \@@_pgf_rect_node:nnnnn
7952 { \@@_env: - #1 - #2 - block }
7953 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7954 \str_if_empty:NF \l_@@_block_name_str
7955 {
7956   \pgfnodealias
7957   { \@@_env: - \l_@@_block_name_str }
7958   { \@@_env: - #1 - #2 - block }
7959   \str_if_empty:NF \l_@@_name_str
7960   {
7961     \pgfnodealias
7962     { \l_@@_name_str - \l_@@_block_name_str }
7963     { \@@_env: - #1 - #2 - block }
7964   }
7965 }

```

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.

```

7966 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7967 {
7968   \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.

```

7969     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7970     {

```

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.

```

7971         \cs_if_exist:cT
7972         { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7973         {
7974             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7975             {
7976                 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7977                 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7978             }
7979         }
7980     }

```

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.

```

7981     \dim_compare:nNnT { \l_tmpb_dim } = { \c_max_dim }
7982     {
7983         \@@_qpoint:n { col - #2 }
7984         \dim_set_eq:NN \l_tmpb_dim \pgf@x
7985     }
7986     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7987     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7988     {
7989         \cs_if_exist:cT
7990         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7991         {
7992             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7993             {
7994                 \pgfpointanchor
7995                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7996                 { east }
7997                 \dim_set:Nn \l_@@_tmpd_dim
7998                 { \dim_max:nn { \l_@@_tmpd_dim } { \pgf@x } }
7999             }
8000         }
8001     }
8002     \dim_compare:nNnT { \l_@@_tmpd_dim } = { - \c_max_dim }
8003     {
8004         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8005         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8006     }
8007     \@@_pgf_rect_node:nnnn
8008     { \@@_env: - #1 - #2 - block - short }
8009     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8010 }

```

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.

```

8011     \bool_if:NT \l_@@_medium_nodes_bool
8012     {
8013         \@@_pgf_rect_node:nnn
8014         { \@@_env: - #1 - #2 - block - medium }
8015         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
8016         {
8017             \pgfpointanchor
8018             { \@@_env:
8019               - \int_use:N \l_@@_last_row_int
8020               - \int_use:N \l_@@_last_col_int - medium
8021             }

```

```

8022         { south~east }
8023     }
8024 }
8025 \endpgfpicture
8026

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand `&`.

```

8027 \bool_if:NTF \l_@@_ampersand_bool
8028 {
8029     \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
8030     \int_zero_new:N \l_@@_split_int
8031     \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }

```

The following counters will be used to send information to `\cellcolor` if the user uses that command in a subcell.

```

8032     \int_zero_new:N \l_@@_first_row_int
8033     \int_zero_new:N \l_@@_first_col_int
8034     \int_zero_new:N \l_@@_last_row_int
8035     \int_zero_new:N \l_@@_last_col_int
8036     \int_set:Nn \l_@@_first_row_int { #1 }
8037     \int_set:Nn \l_@@_first_col_int { #2 }
8038     \int_set:Nn \l_@@_last_row_int { #3 }
8039     \int_set:Nn \l_@@_last_col_int { #4 }

8040     \pgfpicture
8041     \pgfrememberpicturerepositiononpagetrue
8042     \pgf@relevantforpicturesizefalse
8043     \@@_qpoint:n { row - #1 }
8044     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8045     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8046     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
8047     \@@_qpoint:n { col - #2 }
8048     \dim_set_eq:NN \l_tmpa_dim \pgf@x
8049     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8050     \dim_set:Nn \l_tmpb_dim
8051     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
8052     \bool_lazy_or:nnT
8053     { \l_@@_vlines_block_bool }
8054     { \str_if_eq_p:ee { \l_@@_vlines_clist } { all } }
8055     {
8056         \int_step_inline:nn { \l_@@_split_int - 1 }
8057         {
8058             \pgfpathmoveto
8059             {
8060                 \pgfpoint
8061                 { \l_tmpa_dim + ##1 \l_tmpb_dim }
8062                 \l_@@_tmpc_dim
8063             }
8064             \pgfpathlineto
8065             {
8066                 \pgfpoint
8067                 { \l_tmpa_dim + ##1 \l_tmpb_dim }
8068                 \l_@@_tmpd_dim
8069             }
8070             \CT@arc@
8071             \pgfsetlinewidth { 1.1 \arrayrulewidth }
8072             \pgfsetrectcap
8073             \pgfusepathqstroke
8074         }
8075     }
8076     \cs_set_eq:NN \cellcolor \@@_subcellcolor
8077     \int_zero_new:N \l_@@_split_i_int
8078     \str_if_eq:eeTF { \l_@@_vpos_block_str } { T }

```

```

8079 {
8080   \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
8081   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8082 }
8083 {
8084   \str_if_eq:eeTF { \l_@@_vpos_block_str } { B }
8085   {
8086     \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
8087     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8088   }
8089   {
8090     \bool_lazy_or:nnTF
8091     { \int_compare_p:nNn { #1 } = { #3 } }
8092     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
8093     {
8094       \@@_qpoint:n { row - #1 - base }
8095       \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8096     }
8097     {
8098       \str_if_eq:eeTF { \l_@@_vpos_block_str } { b }
8099       {
8100         \@@_qpoint:n { row - #3 - base }
8101         \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8102       }
8103       {
8104         \@@_qpoint:n { #1 - #2 - block }
8105         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8106       }
8107     }
8108   }
8109 }
8110 \int_step_inline:nn { \l_@@_split_int }
8111 {
8112   \group_begin:

```

The counter `\l_@@_split_i_int` is only for the command `\@@_subcellcolor`.

```

8113   \int_set:Nn \l_@@_split_i_int { ##1 }
8114   \dim_set:Nn \col@sep
8115   { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
8116   \pgftransformshift
8117   {
8118     \pgfpoint
8119     {
8120       \l_tmpa_dim + ##1 \l_tmpb_dim -
8121       \str_case:on \l_@@_hpos_block_str
8122       {
8123         l { \l_tmpb_dim + \col@sep }
8124         c { 0.5 \l_tmpb_dim }
8125         r { \col@sep }
8126       }
8127     }
8128     { \l_@@_tmpc_dim }
8129   }
8130   \pgfset { inner~sep = \c_zero_dim }
8131   \pgfnode
8132   { rectangle }
8133   {
8134     \str_if_eq:eeTF { \l_@@_vpos_block_str } { T }
8135     {
8136       \str_case:on \l_@@_hpos_block_str
8137       {
8138         l { north-west }
8139         c { north }
8140         r { north-east }

```

```

8141     }
8142   }
8143   {
8144     \str_if_eq:eeTF { \l_@@_vpos_block_str } { B }
8145     {
8146       \str_case:on \l_@@_hpos_block_str
8147       {
8148         l { south~west }
8149         c { south }
8150         r { south~east }
8151       }
8152     }
8153     {
8154       \bool_lazy_any:nTF
8155       {
8156         { \int_compare_p:nNn { #1 } = { #3 } }
8157         { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
8158         { \str_if_eq_p:ee { \l_@@_vpos_block_str } { b } }
8159       }
8160       {
8161         \str_case:on \l_@@_hpos_block_str
8162         {
8163           l { base~west }
8164           c { base }
8165           r { base~east }
8166         }
8167       }
8168       {
8169         \str_case:on \l_@@_hpos_block_str
8170         {
8171           l { west }
8172           c { center }
8173           r { east }
8174         }
8175       }
8176     }
8177   }
8178 }
8179 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8180 \group_end:
8181 }
8182 \endpgfpicture
8183 }

```

Now the case where there is no ampersand & in the content of the block.

```

8184 {
8185   \bool_if:NTF \l_@@_p_block_bool
8186   {

```

When the final user has used the key p, we have to compute the width.

```

8187   \pgfpicture
8188   \pgfrememberpicturepositiononpagetrue
8189   \pgf@relevantforpicturesizefalse
8190   \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8191   {
8192     \@@_qpoint:n { col - #2 }
8193     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8194     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8195   }
8196   {
8197     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8198     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8199     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8200   }

```

```

8201         \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8202     \endpgfpicture
8203     \hbox_set:Nn \l_@@_cell_box
8204     {
8205         \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8206             { \g_tmpb_dim }
8207         \str_case:on \l_@@_hpos_block_str
8208             { c \centering r \raggedleft l \raggedright j { } }
8209             #6
8210         \end { minipage }
8211     }
8212 }
8213 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
8214 \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.

```

8215     \pgfpicture
8216     \pgfrememberpicturepositiononpagetrue
8217     \pgf@relevantforpicturesizefalse
8218     \bool_lazy_any:nTF
8219     {
8220         { \str_if_empty_p:N \l_@@_vpos_block_str }
8221         { \str_if_eq_p:ee { \l_@@_vpos_block_str } { c } }
8222         { \str_if_eq_p:ee { \l_@@_vpos_block_str } { T } }
8223         { \str_if_eq_p:ee { \l_@@_vpos_block_str } { B } }
8224     }
8225     {

```

If we are in the “first column”, we must put the block as if it was with the key `r`.

```

8226         \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`.

```

8227         \bool_if:nT \g_@@_last_col_found_bool
8228         {
8229             \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
8230             { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8231         }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8232         \tl_set:Ne \l_tmpa_tl
8233         {
8234             \str_case:on \l_@@_vpos_block_str
8235             {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8236             { } {
8237                 \str_case:on \l_@@_hpos_block_str
8238                 {
8239                     c { center }
8240                     l { west }
8241                     r { east }
8242                     j { center }
8243                 }
8244             }
8245             c {
8246                 \str_case:on \l_@@_hpos_block_str
8247                 {
8248                     c { center }
8249                     l { west }
8250                     r { east }
8251                     j { center }

```



```

8252         }
8253     }
8254     T {
8255         \str_case:on \l_@@_hpos_block_str
8256         {
8257             c { north }
8258             l { north~west }
8259             r { north~east }
8260             j { north }
8261         }
8262     }
8263 }
8264 }
8265 B {
8266     \str_case:on \l_@@_hpos_block_str
8267     {
8268         c { south }
8269         l { south~west }
8270         r { south~east }
8271         j { south }
8272     }
8273 }
8274 }
8275 }
8276 }
8277 \pgftransformshift
8278 {
8279     \pgfpointanchor
8280     {
8281         \@@_env: - #1 - #2 - block
8282         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8283     }
8284     { \l_tmpa_tl }
8285 }
8286 \pgfset { inner~sep = \c_zero_dim }
8287 \pgfnode
8288 { rectangle }
8289 { \l_tmpa_tl }
8290 { \box_use_drop:N \l_@@_cell_box } { } { }
8291 }

```

End of the case when `\l_@@_vpos_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```

8292 {
8293     \pgfextracty \l_tmpa_dim
8294     {
8295         \@@_qpoint:n
8296         {
8297             row - \str_if_eq:eeTF { \l_@@_vpos_block_str } { b } { #3 } { #1 }
8298             - base
8299         }
8300     }
8301     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

8302 \pgfpointanchor
8303 {
8304     \@@_env: - #1 - #2 - block
8305     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8306 }
8307 {
8308     \str_case:on \l_@@_hpos_block_str
8309     {
8310         c { center }

```

```

8311         l { west }
8312         r { east }
8313         j { center }
8314     }
8315 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8316     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8317     \pgfset { inner~sep = \c_zero_dim }
8318     \pgfnode
8319     { rectangle }
8320     {
8321         \str_case:on \l_@@_hpos_block_str
8322         {
8323             c { base }
8324             l { base~west }
8325             r { base~east }
8326             j { base }
8327         }
8328     }
8329     { \box_use_drop:N \l_@@_cell_box } { } { }
8330 }
8331 \endpgfpicture
8332 }
8333 \group_end:
8334 }
8335 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }

```

The following command adds the value of `\l_@@_opacity_tl` (if not empty) to the specification of color set in `\l_@@_fill_tl` (the information of opacity is added in between square brackets before the color itself).

```

8336 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8337 {
8338     \tl_if_empty:NF \l_@@_opacity_tl
8339     {
8340         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8341             {
8342                 \tl_set:Nc \l_@@_fill_tl
8343                 {
8344                     [ opacity = \l_@@_opacity_tl ,
8345                     \tl_tail:o \l_@@_fill_tl
8346                 }
8347             }
8348             {
8349                 \tl_set:Nc \l_@@_fill_tl
8350                 { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8351             }
8352         }
8353     }

```

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).

```

8354 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8355 {
8356     \group_begin:
8357     \tl_clear:N \l_@@_draw_tl
8358     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8359     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8360     \pgfpicture
8361     \pgfrememberpicturepositiononpagetrue
8362     \pgf@relevantforpicturesizefalse

```

```

8363 \tl_if_empty:NF \l_@@_draw_tl
8364 {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8365 \tl_if_eq:NnTF \l_@@_draw_tl { default }
8366 { \CT@arc@ }
8367 { \@@_color:o \l_@@_draw_tl }
8368 }
8369 \pgfsetcornersarced
8370 {
8371 \pgfpoint
8372 { \l_@@_rounded_corners_dim }
8373 { \l_@@_rounded_corners_dim }
8374 }
8375 \@@_cut_on_hyphen:w #2 \q_stop
8376 \int_compare:nNnF { \l_tmpa_tl } > { \c@iRow }
8377 {
8378 \int_compare:nNnF { \l_tmpb_tl } > { \c@jCol }
8379 {
8380 \@@_qpoint:n { row - \l_tmpa_tl }
8381 \dim_set_eq:NN \l_tmpb_dim \pgf@y
8382 \@@_qpoint:n { col - \l_tmpb_tl }
8383 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8384 \@@_cut_on_hyphen:w #3 \q_stop
8385 \int_compare:nNnT { \l_tmpa_tl } > { \c@iRow }
8386 { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8387 \int_compare:nNnT { \l_tmpb_tl } > { \c@jCol }
8388 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8389 \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8390 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8391 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8392 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8393 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8394 \pgfpathrectanglecorners
8395 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8396 { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8397 \dim_compare:nNnTF { \l_@@_rounded_corners_dim } = { \c_zero_dim }
8398 { \pgfusepathqstroke }
8399 { \pgfusepath { stroke } }
8400 }
8401 }
8402 \endpgfpicture
8403 \group_end:
8404 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8405 \keys_define:nn { nicematrix / BlockStroke }
8406 {
8407 color .tl_set:N = \l_@@_draw_tl ,
8408 draw .code:n =
8409 \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8410 draw .default:n = default ,
8411 line-width .dim_set:N = \l_@@_line_width_dim ,
8412 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8413 rounded-corners .default:n = 4 pt
8414 }

```

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).

```

8415 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8416 {

```

```

8417 \group_begin:
8418 \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8419 \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8420 \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8421 \@@_cut_on_hyphen:w #2 \q_stop
8422 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8423 \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8424 \@@_cut_on_hyphen:w #3 \q_stop
8425 \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8426 \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8427 \int_step_inline:nnn { \l_@@_tmpd_tl } { \l_tmpb_tl }
8428 {
8429   \use:e
8430   {
8431     \@@_vline:n
8432     {
8433       position = ##1 ,
8434       start = \l_@@_tmpc_tl ,
8435       end = \int_eval:n { \l_tmpa_tl - 1 } ,
8436       total-width = \dim_use:N \l_@@_line_width_dim
8437     }
8438   }
8439 }
8440 \group_end:
8441 }

8442 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8443 {
8444   \group_begin:
8445   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8446   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8447   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8448   \@@_cut_on_hyphen:w #2 \q_stop
8449   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8450   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8451   \@@_cut_on_hyphen:w #3 \q_stop
8452   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8453   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8454   \int_step_inline:nnn { \l_@@_tmpc_tl } { \l_tmpa_tl }
8455   {
8456     \use:e
8457     {
8458       \@@_hline:n
8459       {
8460         position = ##1 ,
8461         start = \l_@@_tmpd_tl ,
8462         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8463         total-width = \dim_use:N \l_@@_line_width_dim
8464       }
8465     }
8466   }
8467   \group_end:
8468 }

```

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).

```

8469 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8470 {
8471   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8472   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8473   \dim_compare:nNnTF { \l_@@_rounded_corners_dim } > { \c_zero_dim }
8474   { \@@_error:n { borders~forbidden } }

```

```

8475 {
8476   \tl_clear_new:N \l_@@_borders_tikz_tl
8477   \keys_set:no
8478     { nicematrix / OnlyForTikzInBorders }
8479   \l_@@_borders_clist
8480   \@@_cut_on_hyphen:w #2 \q_stop
8481   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8482   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8483   \@@_cut_on_hyphen:w #3 \q_stop
8484   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8485   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8486   \@@_stroke_borders_block_i:
8487 }
8488 }

8489 \hook_gput_code:nnn { begindocument } { . }
8490 {
8491   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8492   {
8493     \c_@@_pgfortikzpicture_tl
8494     \@@_stroke_borders_block_ii:
8495     \c_@@_endpgfortikzpicture_tl
8496   }
8497 }

8498 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8499 {
8500   \pgfrememberpicturepositiononpagetrue
8501   \pgf@relevantforpicturesizefalse
8502   \CT@arc@
8503   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8504   \clist_if_in:NnT \l_@@_borders_clist { right }
8505     { \@@_stroke_vertical:n \l_tmpb_tl }
8506   \clist_if_in:NnT \l_@@_borders_clist { left }
8507     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8508   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8509     { \@@_stroke_horizontal:n \l_tmpa_tl }
8510   \clist_if_in:NnT \l_@@_borders_clist { top }
8511     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8512 }

8513 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8514 {
8515   tikz .code:n =
8516     \cs_if_exist:NTF \tikzpicture
8517       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8518       { \@@_error:n { tikz~in~borders~without~tikz } } ,
8519   tikz .value_required:n = true ,
8520   top .code:n = ,
8521   bottom .code:n = ,
8522   left .code:n = ,
8523   right .code:n = ,
8524   unknown .code:n = \@@_error:n { bad~border }
8525 }

```

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).

```

8526 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8527 {
8528   \@@_qpoint:n \l_@@_tmpc_tl
8529   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8530   \@@_qpoint:n \l_tmpa_tl
8531   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8532   \@@_qpoint:n { #1 }
8533   \tl_if_empty:NTF \l_@@_borders_tikz_tl

```

```

8534 {
8535   \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8536   \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8537   \pgfusepathqstroke
8538 }
8539 {
8540   \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8541   ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8542 }
8543 }

```

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).

```

8544 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8545 {
8546   \@@_qpoint:n \l_@@_tmpd_tl
8547   \clist_if_in:NnTF \l_@@_borders_clist { left }
8548   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8549   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8550   \@@_qpoint:n \l_tmpb_tl
8551   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8552   \@@_qpoint:n { #1 }
8553   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8554   {
8555     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8556     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8557     \pgfusepathqstroke
8558   }
8559   {
8560     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8561     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8562   }
8563 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

8564 \keys_define:nn { nicematrix / BlockBorders }
8565 {
8566   borders .clist_set:N = \l_@@_borders_clist ,
8567   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8568   rounded-corners .default:n = 4 pt ,
8569   line-width .dim_set:N = \l_@@_line_width_dim
8570 }

```

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.

```

8571 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8572 {
8573   \begin { tikzpicture }
8574   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because #5 is a list of lists.

```

8575   \clist_map_inline:nn { #1 }
8576   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8577 \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8578 \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8579 (
8580 [
8581 xshift = \dim_use:N \l_@@_offset_dim ,
8582 yshift = - \dim_use:N \l_@@_offset_dim
8583 ]
8584 #2 -| #3
8585 )
8586 rectangle
8587 (
8588 [
8589 xshift = - \dim_use:N \l_@@_offset_dim ,
8590 yshift = \dim_use:N \l_@@_offset_dim
8591 ]
8592 \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8593 ) ;
8594 }
8595 \end { tikzpicture }
8596 }
8597 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

8598 \keys_define:nn { nicematrix / SpecialOffset }
8599 { 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.

```

8600 \cs_new_protected:Npn \@@_NullBlock:
8601 { \@@_collect_options:n { \@@_NullBlock_i: } }
8602 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8603 { }

```

The following command will be linked to `\cellcolor` in the sub-cells of a block which contains ampersands (&). Of course, `&-in-blocks` must be in force.

```

8604 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
8605 {
8606 \tl_gput_right:Ne \g_@@_pre_code_before_tl
8607 {

```

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`).

```

8608 \@@_subcellcolor:nnnnnnn
8609 {
8610 \tl_if_blank:nTF { #1 }
8611 { { \exp_not:n { #2 } } }
8612 { [ #1 ] { \exp_not:n { #2 } } }
8613 }
8614 { \int_use:N \l_@@_first_row_int } % first row of the block
8615 { \int_use:N \l_@@_first_col_int } % first column of the block
8616 { \int_use:N \l_@@_last_row_int } % last row of the block
8617 { \int_use:N \l_@@_last_col_int } % last column of the block
8618 { \int_use:N \l_@@_split_int }
8619 { \int_use:N \l_@@_split_i_int }
8620 }
8621 \ignorespaces
8622 }

8623 \cs_new_protected:Npn \@@_subcellcolor:nnnnnn #1 #2 #3 #4 #5 #6 #7
8624 {
8625 \@@_color_opacity: #1

```

```

8626 \pgfpicture
8627 \pgf@relevantforpicturesizefalse
8628 \@@_qpoint:n { col - #3 }
8629 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8630 \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
8631 \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
8632 \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
8633 \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
8634 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8635 \@@_qpoint:n { row - #2 }
8636 \pgfpathrectanglecorners
8637 { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } { \l_@@_tmpc_dim } }
8638 { \pgfpoint { \l_tmpb_dim } { \pgf@y } }
8639 \pgfusepathqfill
8640 \endpgfpicture
8641 }

```

27 How to draw the dotted lines transparently

```

8642 \cs_set_protected:Npn \@@_renew_matrix:
8643 {
8644   \RenewDocumentEnvironment { pmatrix } { } {
8645     { \pNiceMatrix }
8646     { \endpNiceMatrix }
8647   \RenewDocumentEnvironment { vmatrix } { } {
8648     { \vNiceMatrix }
8649     { \endvNiceMatrix }
8650   \RenewDocumentEnvironment { Vmatrix } { } {
8651     { \VNiceMatrix }
8652     { \endVNiceMatrix }
8653   \RenewDocumentEnvironment { bmatrix } { } {
8654     { \bNiceMatrix }
8655     { \endbNiceMatrix }
8656   \RenewDocumentEnvironment { Bmatrix } { } {
8657     { \BNiceMatrix }
8658     { \endBNiceMatrix }
8659   }

```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8660 \keys_define:nn { nicematrix / Auto }
8661 {
8662   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8663   columns-type .value_required:n = true ,
8664   l .meta:n = { columns-type = l } ,
8665   r .meta:n = { columns-type = r } ,
8666   c .meta:n = { columns-type = c } ,
8667   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8668   delimiters / color .value_required:n = true ,
8669   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8670   delimiters / max-width .default:n = true ,
8671   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8672   delimiters .value_required:n = true ,
8673   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8674   rounded-corners .default:n = 4 pt
8675 }

```



```

8676 \NewDocumentCommand \AutoNiceMatrixWithDelims
8677 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8678 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8679 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8680 {

```

The group is for the protection of the keys.

```

8681 \group_begin:
8682 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8683 \use:e
8684 {
8685 \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8686 { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8687 [ \exp_not:o \l_tmpa_tl ]
8688 }
8689 \int_if_zero:nT { \l_@@_first_row_int }
8690 {
8691 \int_if_zero:nT { \l_@@_first_col_int } { & }
8692 \prg_replicate:nn { #4 - 1 } { & }
8693 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8694 }
8695 \prg_replicate:nn { #3 }
8696 {
8697 \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).

```

8698 \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8699 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8700 }
8701 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
8702 {
8703 \int_if_zero:nT { \l_@@_first_col_int } { & }
8704 \prg_replicate:nn { #4 - 1 } { & }
8705 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8706 }
8707 \end { NiceArrayWithDelims }
8708 \group_end:
8709 }

8710 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
8711 {
8712 \cs_set_protected:cpn { #1 AutoNiceMatrix }
8713 {
8714 \bool_gset_true:N \g_@@_delims_bool
8715 \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8716 \AutoNiceMatrixWithDelims { #2 } { #3 }
8717 }
8718 }

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8719 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8720 {
8721 \group_begin:
8722 \bool_gset_false:N \g_@@_delims_bool
8723 \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8724 \group_end:
8725 }

```

29 The redefinition of the command `\dotfill`

```

8726 \cs_set_eq:NN \@@_old_dotfill: \dotfill
8727 \cs_new_protected:Npn \@@_dotfill:
8728 {

```

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}`).

```

8729 \@@_old_dotfill:
8730 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8731 }

```

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`.

```

8732 \cs_new_protected:Npn \@@_dotfill_i:
8733 {
8734   \dim_compare:nNtT { \box_wd:N \l_@@_cell_box } = { \c_zero_dim }
8735   { \@@_old_dotfill: }
8736 }

```

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.

```

8737 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8738 {
8739   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8740   {
8741     \@@_actually_diagbox:nnnnnn
8742     { \int_use:N \c@iRow }
8743     { \int_use:N \c@jCol }
8744     { \int_use:N \c@iRow }
8745     { \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.

```

8746   { \g_@@_row_style_tl \exp_not:n { #1 } }
8747   { \g_@@_row_style_tl \exp_not:n { #2 } }
8748 }

```

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`.

```

8749 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8750 {
8751   { \int_use:N \c@iRow }
8752   { \int_use:N \c@jCol }
8753   { \int_use:N \c@iRow }
8754   { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8755   { }
8756 }
8757 }

```

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.

```

8758 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8759 {
8760   \pgfpicture
8761   \pgf@relevantforpicturesizefalse
8762   \pgfrememberpicturepositiononpagetrue
8763   \@@_qpoint:n { row - #1 }
8764   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8765   \@@_qpoint:n { col - #2 }
8766   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8767   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8768   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8769   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8770   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8771   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8772   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8773   {

```

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.

```

8774   \CT@arc@
8775   \pgfsetroundcap
8776   \pgfusepathqstroke
8777 }
8778 \pgfset { inner~sep = 1 pt }
8779 \pgfscope
8780 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8781 \pgfnode { rectangle } { south~west }
8782 {
8783   \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8784   \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8785   \end { minipage }
8786 }
8787 { }
8788 { }
8789 \endpgfscope
8790 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8791 \pgfnode { rectangle } { north~east }
8792 {
8793   \begin { minipage } { 20 cm }
8794   \raggedleft
8795   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8796   \end { minipage }
8797 }
8798 { }
8799 { }
8800 \endpgfpicture
8801 }

```

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. 87.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
8802 \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 `\`.

```
8803 \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`.

```
8804 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8805 {
8806   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8807   \@@_CodeAfter_iv:n
8808 }
```

We catch the argument of the command `\end` (in `#1`).

```
8809 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8810 {
```

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.

```
8811   \str_if_eq:eeTF { \currenvir } { #1 }
8812   { \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`.

```
8813   {
8814     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8815     \@@_CodeAfter_ii:n
8816   }
8817 }
```

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.

```
8818 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8819 {
8820   \pgfpicture
8821   \pgfrememberpicturepositiononpagetrue
8822   \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.

```
8823   \@@_qpoint:n { row - 1 }
8824   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8825   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8826   \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).

```

8827 \bool_if:nTF { #3 }
8828 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8829 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8830 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8831 {
8832   \cs_if_exist:cT
8833   { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8834   {
8835     \pgfpointanchor
8836     { \@@_env: - ##1 - #2 }
8837     { \bool_if:nTF { #3 } { west } { east } }
8838     \dim_set:Nn \l_tmpa_dim
8839     {
8840       \bool_if:nTF { #3 }
8841       { \dim_min:nn }
8842       { \dim_max:nn }
8843       \l_tmpa_dim
8844       { \pgf@x }
8845     }
8846   }
8847 }

```

Now we can put the delimiter with a node of PGF.

```

8848 \pgfset { inner~sep = \c_zero_dim }
8849 \dim_zero:N \nulldelimiterspace
8850 \pgftransformshift
8851 {
8852   \pgfpoint
8853   { \l_tmpa_dim }
8854   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8855 }
8856 \pgfnode
8857 { rectangle }
8858 { \bool_if:nTF { #3 } { east } { west } }
8859 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8860 \nullfont
8861 $ % $
8862 \@@_color:o \l_@@_delimiters_color_tl
8863 \bool_if:nTF { #3 } { \left #1 } { \left . }
8864 \vcenter
8865 {
8866   \nullfont
8867   \hrule \@height
8868   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8869   \@depth \c_zero_dim
8870   \@width \c_zero_dim
8871 }
8872 \bool_if:nTF { #3 } { \right . } { \right #1 }
8873 $ % $
8874 }
8875 { }
8876 { }
8877 \endpgfpicture
8878 }

```

33 The command `\SubMatrix`

```

8879 \keys_define:nn { nicematrix / sub-matrix }
8880 {
8881   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8882   extra-height .value_required:n = true ,
8883   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8884   left-xshift .value_required:n = true ,
8885   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8886   right-xshift .value_required:n = true ,
8887   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8888   xshift .value_required:n = true ,
8889   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8890   delimiters / color .value_required:n = true ,
8891   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8892   slim .default:n = true ,
8893   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8894   hlines .default:n = all ,
8895   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8896   vlines .default:n = all ,
8897   hvlines .meta:n = { hlines, vlines } ,
8898   hvlines .value_forbidden:n = true
8899 }
8900 \keys_define:nn { nicematrix }
8901 {
8902   SubMatrix .inherit:n = nicematrix / sub-matrix ,
8903   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8904   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8905   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8906 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8907 \keys_define:nn { nicematrix / SubMatrix }
8908 {
8909   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8910   delimiters / color .value_required:n = true ,
8911   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8912   hlines .default:n = all ,
8913   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8914   vlines .default:n = all ,
8915   hvlines .meta:n = { hlines, vlines } ,
8916   hvlines .value_forbidden:n = true ,
8917   name .code:n =
8918     \tl_if_empty:nTF { #1 }
8919     { \@@_error:n { Invalid-name } }
8920     {
8921       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8922       {
8923         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8924         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8925         {
8926           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8927           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8928         }
8929       }
8930       { \@@_error:n { Invalid-name } }
8931     } ,
8932   name .value_required:n = true ,
8933   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8934   rules .value_required:n = true ,
8935   code .tl_set:N = \l_@@_code_tl ,
8936   code .value_required:n = true ,
8937   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
8938 }

```

```

8939 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
8940 {
8941   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8942   {
8943     \SubMatrix { #1 } { #2 } { #3 } { #4 }
8944     [
8945       delimiters / color = \l_@@_delimiters_color_tl ,
8946       hlines = \l_@@_submatrix_hlines_clist ,
8947       vlines = \l_@@_submatrix_vlines_clist ,
8948       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8949       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8950       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8951       slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8952       #5
8953     ]
8954   }
8955   \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8956   \ignorespaces
8957 }
8958 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8959 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8960 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8961 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8962 {
8963   \seq_gput_right:Ne \g_@@_submatrix_seq
8964   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8965   { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8966   { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8967   { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8968   { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8969 }
8970 }

```

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).

```

8971 \NewDocumentCommand \@@_compute_i_j:nn
8972 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8973 { \@@_compute_i_j:nnnn #1 #2 }
8974 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8975 {
8976   \def \l_@@_first_i_tl { #1 }
8977   \def \l_@@_first_j_tl { #2 }
8978   \def \l_@@_last_i_tl { #3 }
8979   \def \l_@@_last_j_tl { #4 }
8980   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8981     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8982   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8983     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8984   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8985     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8986   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8987     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8988 }

```

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`.

```

8989 \hook_gput_code:nnn { begindocument } { . }
8990 {
8991   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
8992   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
8993     { \@@_sub_matrix:nnnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
8994 }
8995 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8996 {
8997   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8998   \@@_compute_i_j:nn { #2 } { #3 }
8999   \int_compare:nNt { \l_@@_first_i_tl } = { \l_@@_last_i_tl }
9000   { \def \arraystretch { 1 } }
9001   \bool_lazy_or:nnTF
9002     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9003     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9004     { \@@_error:nn { Construct~too-large } { \SubMatrix } }
9005   {
9006     \str_clear_new:N \l_@@_submatrix_name_str
9007     \keys_set:nn { nicematrix / SubMatrix } { #5 }
9008     \pgfpicture
9009     \pgfrememberpicturepositiononpagetrue
9010     \pgf@relevantforpicturesizefalse
9011     \pgfset { inner~sep = \c_zero_dim }
9012     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9013     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curryfication.

```

9014   \bool_if:NTF \l_@@_submatrix_slim_bool
9015     { \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl } }
9016     { \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int } }
9017   {
9018     \cs_if_exist:cT
9019       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9020     {
9021       \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9022       \dim_compare:nNt { \pgf@x } < { \l_@@_x_initial_dim }
9023       { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9024     }
9025     \cs_if_exist:cT
9026       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9027     {
9028       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9029       \dim_compare:nNt { \pgf@x } > { \l_@@_x_final_dim }
9030       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9031     }
9032   }
9033   \dim_compare:nNtTF { \l_@@_x_initial_dim } = { \c_max_dim }
9034     { \@@_error:nn { Impossible~delimiter } { left } }
9035   {
9036     \dim_compare:nNtTF { \l_@@_x_final_dim } = { - \c_max_dim }

```



```

9037         { \@@_error:nn { Impossible-delimiter } { right } }
9038         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
9039     }
9040     \endpgfpicture
9041 }
9042 \group_end:
9043 \ignorespaces
9044 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

9045 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
9046 {
9047     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
9048     \dim_set:Nn \l_@@_y_initial_dim
9049     {
9050         \fp_to_dim:n
9051         {
9052             \pgf@y
9053             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
9054         }
9055     }
9056     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
9057     \dim_set:Nn \l_@@_y_final_dim
9058     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
9059     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
9060     {
9061         \cs_if_exist:cT
9062         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
9063         {
9064             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
9065             \dim_set:Nn \l_@@_y_initial_dim
9066             { \dim_max:nn { \l_@@_y_initial_dim } { \pgf@y } }
9067         }
9068         \cs_if_exist:cT
9069         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
9070         {
9071             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
9072             \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
9073             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
9074         }
9075     }
9076     \dim_set:Nn \l_tmpa_dim
9077     {
9078         \l_@@_y_initial_dim - \l_@@_y_final_dim +
9079         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
9080     }
9081     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

9082     \group_begin:
9083     \pgfsetlinewidth { 1.1 \arrayrulewidth }
9084     \@@_set_CTarc:o \l_@@_rules_color_tl
9085     \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`.

```

9086     \seq_map_inline:Nn \g_@@_cols_vlism_seq
9087     {
9088         \int_compare:nNnT { \l_@@_first_j_tl } < { ##1 }
9089         {
9090             \int_compare:nNnT
9091             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }

```

```
9092         {
```

First, we extract the value of the abscissa of the rule we have to draw.

```
9093             \@_qpoint:n { col - ##1 }
9094             \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9095             \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9096             \pgfusepathqstroke
9097         }
9098     }
9099 }
```

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.

```
9100 \str_if_eq:eeTF { \l_@@_submatrix_vlines_clist } { all }
9101 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
9102 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
9103 {
9104     \bool_lazy_and:nnTF
9105     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
9106     {
9107         \int_compare_p:nNn
9108         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
9109     {
9110         \@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
9111         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9112         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9113         \pgfusepathqstroke
9114     }
9115     { \@_error:nmn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
9116 }
```

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.

```
9117 \str_if_eq:eeTF { \l_@@_submatrix_hlines_clist } { all }
9118 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
9119 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
9120 {
9121     \bool_lazy_and:nnTF
9122     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
9123     {
9124         \int_compare_p:nNn
9125         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
9126     {
9127         \@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }
```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```
9128     \group_begin:
```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```
9129     \dim_set:Nn \l_tmpa_dim
9130     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9131     \str_case:nn { #1 }
9132     {
9133         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9134         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
9135         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9136     }
9137     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```
9138     \dim_set:Nn \l_tmpb_dim
9139     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9140     \str_case:nn { #2 }
9141     {
9142         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
```

```

9143         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
9144         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9145     }
9146     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9147     \pgfusepathqstroke
9148     \group_end:
9149 }
9150 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { #1 } }
9151 }

```

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).

```

9152 \str_if_empty:NF \l_@@_submatrix_name_str
9153 {
9154     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
9155     \l_@@_x_initial_dim \l_@@_y_initial_dim
9156     \l_@@_x_final_dim \l_@@_y_final_dim
9157 }
9158 \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`.

```

9159 \begin { pgfscope }
9160 \pgftransformshift
9161 {
9162     \pgfpoint
9163     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9164     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9165 }
9166 \str_if_empty:NTF \l_@@_submatrix_name_str
9167 { \@@_node_left:nn #1 { } }
9168 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9169 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9170 \pgftransformshift
9171 {
9172     \pgfpoint
9173     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9174     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9175 }
9176 \str_if_empty:NTF \l_@@_submatrix_name_str
9177 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9178 {
9179     \@@_node_right:nnnn #2
9180     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9181 }

```

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`.

```

9182 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9183 \flag_clear_new:N \l_@@_code_flag
9184 \l_@@_code_tl
9185 }

```

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`.

```

9186 \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.

```

9187 \cs_new:Npn \@@_pgfpointanchor:n #1
9188 { \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`).

```

9189 \cs_new:Npn \@@_pgfpointanchor_i:n #1
9190 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }

9191 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9192 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9193 \str_if_empty:nTF { #1 }

```

First, when the name of the name begins with `\tikz@pp@name`.

```

9194 { \@@_pgfpointanchor_iv:w #2 }

```

And now, when there is no `\tikz@pp@name`.

```

9195 { \@@_pgfpointanchor_ii:n { #1 } }
9196 }

```

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`).

```

9197 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
9198 { \@@_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`.

```

9199 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

9200 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
9201 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9202 \str_if_empty:nTF { #2 }

```

First the case where the argument does *not* contain an hyphen.

```

9203 { \@@_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`).

```

9204 { \@@_pgfpointanchor_iii:w { #1 } #2 }
9205 }

```

The following function is for the case when the name contains an hyphen.

```

9206 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9207 {

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9208 \@@_env:
9209 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9210 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9211 }

```

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.

```

9212 \tl_const:Nn \c_@@_integers_alist_tl
9213 {
9214   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9215   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9216   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9217   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9218 }

9219 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9220 {

```

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`.

```

9221   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
9222   {
9223     \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9224   \@@_env: -
9225   \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9226   { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9227   { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9228 }
9229 {
9230   \str_if_eq:eeTF { #1 } { last }
9231   {
9232     \flag_raise:N \l_@@_code_flag
9233     \@@_env: -
9234     \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9235     { \int_eval:n { \l_@@_last_i_tl + 1 } }
9236     { \int_eval:n { \l_@@_last_j_tl + 1 } }
9237   }
9238   { #1 }
9239 }
9240 }

```

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`).

```

9241 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9242 {
9243   \pgfnode
9244   { rectangle }
9245   { east }
9246   {
9247     \nullfont
9248     $ % $
9249     \@@_color:o \l_@@_delimiters_color_tl
9250     \left #1
9251     \vcenter
9252     {
9253       \nullfont
9254       \hrule \@height \l_tmpa_dim
9255       \@depth \c_zero_dim

```

```

9256             \@width \c_zero_dim
9257         }
9258         \right .
9259         $ % $
9260     }
9261     { #2 }
9262     { }
9263 }

```

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.

```

9264 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9265 {
9266     \pgfnode
9267     { rectangle }
9268     { west }
9269     {
9270         \nullfont
9271         $ % $
9272         \colorlet { current-color } { . }
9273         \@@_color:o \l_@@_delimiters_color_tl
9274         \left .
9275         \vcenter
9276         {
9277             \nullfont
9278             \hrule \@height \l_tmpa_dim
9279                 \@depth \c_zero_dim
9280                 \@width \c_zero_dim
9281         }
9282         \right #1
9283         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9284         ^ { \color { current-color } \smash { #4 } }
9285         $ % $
9286     }
9287     { #2 }
9288     { }
9289 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9290 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9291 {
9292     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9293     \ignorespaces
9294 }
9295 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9296 {
9297     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9298     \ignorespaces
9299 }
9300 \keys_define:nn { nicematrix / Brace }
9301 {
9302     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9303     left-shorten .default:n = true ,
9304     left-shorten .value_forbidden:n = true ,

```

```

9305   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9306   right-shorten .default:n = true ,
9307   right-shorten .value_forbidden:n = true ,
9308   shorten .meta:n = { left-shorten , right-shorten } ,
9309   shorten .value_forbidden:n = true ,
9310   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9311   yshift .value_required:n = true ,
9312   yshift .initial:n = \c_zero_dim ,
9313   color .tl_set:N = \l_tmpa_tl ,
9314   color .value_required:n = true ,
9315   unknown .code:n =
9316     \@@_unknown_key:nn
9317     { nicematrix / Brace }
9318     { Unknown-key-for-Brace }
9319 }

```

#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*.

```

9320 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9321 {
9322   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9323   \@@_compute_i_j:nn { #1 } { #2 }
9324   \bool_lazy_or:nnTF
9325     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9326     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9327     {
9328       \str_if_eq:eeTF { #5 } { under }
9329       { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9330       { \@@_error:nn { Construct-too-large } { \OverBrace } }
9331     }
9332   {
9333     \tl_clear:N \l_tmpa_tl
9334     \keys_set:nn { nicematrix / Brace } { #4 }
9335     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9336     \pgfpicture
9337     \pgfrememberpicturepositiononpagetrue
9338     \pgf@relevantforpicturesizefalse
9339     \bool_if:NT \l_@@_brace_left_shorten_bool
9340     {
9341       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9342       \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9343       {
9344         \cs_if_exist:cT
9345           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9346           {
9347             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9348
9349             \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9350             { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9351           }
9352       }
9353     }
9354     \bool_lazy_or:nnT
9355       { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9356       { \dim_compare_p:nNn { \l_@@_x_initial_dim } = { \c_max_dim } }
9357     {
9358       \@@_qpoint:n { col - \l_@@_first_j_tl }
9359       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9360     }
9361     \bool_if:NT \l_@@_brace_right_shorten_bool
9362     {

```

```

9363 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9364 \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9365 {
9366   \cs_if_exist:cT
9367   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9368   {
9369     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9370     \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
9371     { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9372   }
9373 }
9374 }
9375 \bool_lazy_or:nnT
9376 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9377 { \dim_compare_p:nNn { \l_@@_x_final_dim } = { - \c_max_dim } }
9378 {
9379   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9380   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9381 }
9382 \pgfset { inner~sep = \c_zero_dim }
9383 \str_if_eq:eeTF { #5 } { under }
9384 { \@@_underbrace_i:n { #3 } }
9385 { \@@_overbrace_i:n { #3 } }
9386 \endpgfpicture
9387 }
9388 \group_end:
9389 }

```

The argument is the text to put above the brace.

```

9390 \cs_new_protected:Npn \@@_overbrace_i:n #1
9391 {
9392   \@@_qpoint:n { row - \l_@@_first_i_tl }
9393   \pgftransformshift
9394   {
9395     \pgfpoint
9396     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9397     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9398   }
9399   \pgfnode
9400   { rectangle }
9401   { south }
9402   {
9403     \vtop
9404     {
9405       \group_begin:
9406       \everycr { }
9407       \halign
9408       {
9409         \hfil ## \hfil \crcr
9410         \bool_if:NTF \l_@@_tabular_bool
9411         { \begin { tabular } { c } #1 \end { tabular } }
9412         { $ \begin { array } { c } #1 \end { array } $ }
9413         \cr
9414         $ % $
9415         \overbrace
9416         {
9417           \hbox_to_wd:nn
9418           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9419           { }
9420         }
9421         $ % $
9422         \cr
9423       }
9424       \group_end:

```



```

9425     }
9426   }
9427   { }
9428   { }
9429 }

```

The argument is the text to put under the brace.

```

9430 \cs_new_protected:Npn \l_@@_underbrace_i:n #1
9431 {
9432   \l_@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9433   \pgftransformshift
9434   {
9435     \pgfpoint
9436     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9437     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9438   }
9439   \pgfnode
9440   { rectangle }
9441   { north }
9442   {
9443     \group_begin:
9444     \everycr { }
9445     \vbox
9446     {
9447       \halign
9448       {
9449         \hfil ## \hfil \crcr
9450         $ % $
9451         \underbrace
9452         {
9453           \hbox_to_wd:nn
9454           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9455           { }
9456         }
9457         $ % $
9458         \cr
9459         \bool_if:NTF \l_@@_tabular_bool
9460         { \begin { tabular } { c } #1 \end { tabular } }
9461         { $ \begin { array } { c } #1 \end { array } $ }
9462         \cr
9463       }
9464     }
9465     \group_end:
9466   }
9467   { }
9468   { }
9469 }

```

35 The commands HBrace et VBrace

The TikZ style `nicematrix/brace` is a TikZ style used to draw the braces created by `\Hbrace` and `\Vbrace`.

We can't load that definition right away because of course, maybe the final user has not yet loaded TikZ (`\Hbrace` and `\Vbrace` are available only when TikZ is loaded and also its library `decorations.pathreplacing`).

```

9470 \AddToHook { package / tikz / after }
9471 {

```

```

9472 \tikzset
9473 {
9474     nicematrix / brace / .style =
9475     {
9476         decoration = { brace , raise = -0.15 em } ,
9477         decorate ,
9478     } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9479     nicematrix / mirrored-brace / .style =
9480     {
9481         nicematrix / brace ,
9482         decoration = mirror ,
9483     }
9484 }
9485 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9486 \keys_define:nn { nicematrix / Hbrace }
9487 {
9488     color .code:n = ,
9489     horizontal-label .code:n = ,
9490     horizontal-labels .code:n = ,
9491     shorten .code:n = ,
9492     shorten-start .code:n = ,
9493     shorten-end .code:n = ,
9494     brace-shift .code:n = ,
9495     unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9496 }

```

Here we need an “fully expandable” command.

```

9497 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9498 {
9499     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9500     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9501     { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9502 }

```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

9503 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9504 {
9505     \int_compare:nNnTF { \c@iRow } < { 2 }
9506     {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9507     \str_if_eq:nnTF { #2 } { * }
9508     {
9509         \bool_set_true:N \l_@@_nullify_dots_bool
9510         \Ldots
9511         [
9512             line-style = nicematrix / brace ,
9513             #1 ,
9514             up =
9515             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9516         ]
9517     }
9518     {
9519         \Hdotsfor
9520         [
9521             line-style = nicematrix / brace ,

```

```

9522         #1 ,
9523         up =
9524         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9525     ]
9526     { #2 }
9527 }
9528 }
9529 {
9530     \str_if_eq:nnTF { #2 } { * }
9531     {
9532         \bool_set_true:N \l_@@_nullify_dots_bool
9533         \Ldots
9534         [
9535             line-style = nicematrix / mirrored-brace ,
9536             #1 ,
9537             down =
9538             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9539         ]
9540     }
9541     {
9542         \Hdotsfor
9543         [
9544             line-style = nicematrix / mirrored-brace ,
9545             #1 ,
9546             down =
9547             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9548         ]
9549         { #2 }
9550     }
9551 }
9552 \keys_set:nn { nicematrix / Hbrace } { #1 }
9553 }

9554 \NewDocumentCommand { \@@_Vbrace } { 0 { } m m }
9555 {
9556     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9557     { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9558     { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9559 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

9560 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
9561 {
9562     \int_compare:nNnTF { \c@jCol } < { 2 }
9563     {
9564         \str_if_eq:nnTF { #2 } { * }
9565         {
9566             \bool_set_true:N \l_@@_nullify_dots_bool
9567             \Vdots
9568             [
9569                 Vbrace ,
9570                 line-style = nicematrix / mirrored-brace ,
9571                 #1 ,
9572                 down =
9573                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9574             ]
9575         }
9576         {
9577             \Vdotsfor
9578             [
9579                 Vbrace ,
9580                 line-style = nicematrix / mirrored-brace ,

```

```

9581         #1 ,
9582         down =
9583         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9584     ]
9585     { #2 }
9586 }
9587 }
9588 {
9589     \str_if_eq:nnTF { #2 } { * }
9590     {
9591         \bool_set_true:N \l_@@_nullify_dots_bool
9592         \Vdots
9593         [
9594             Vbrace ,
9595             line-style = nicematrix / brace ,
9596             #1 ,
9597             up =
9598             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9599         ]
9600     }
9601     {
9602         \Vdotsfor
9603         [
9604             Vbrace ,
9605             line-style = nicematrix / brace ,
9606             #1 ,
9607             up =
9608             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9609         ]
9610         { #2 }
9611     }
9612 }
9613 \keys_set:nn { nicematrix / Hbrace } { #1 }
9614 }

```

36 The command TikzEveryCell

```

9615 \bool_new:N \l_@@_not_empty_bool
9616 \bool_new:N \l_@@_empty_bool
9617
9618 \keys_define:nn { nicematrix / TikzEveryCell }
9619 {
9620     not-empty .code:n =
9621     \bool_lazy_or:nnTF
9622     { \l_@@_in_code_after_bool }
9623     { \g_@@_create_cell_nodes_bool }
9624     { \bool_set_true:N \l_@@_not_empty_bool }
9625     { \@@_error:n { detection~of~empty~cells } } ,
9626     not-empty .value_forbidden:n = true ,
9627     empty .code:n =
9628     \bool_lazy_or:nnTF
9629     { \l_@@_in_code_after_bool }
9630     { \g_@@_create_cell_nodes_bool }
9631     { \bool_set_true:N \l_@@_empty_bool }
9632     { \@@_error:n { detection~of~empty~cells } } ,
9633     empty .value_forbidden:n = true ,
9634     unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
9635 }
9636

```

```

9637
9638 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9639 {
9640   \IfPackageLoadedTF { tikz }
9641   {
9642     \group_begin:
9643     \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.

```

9644     \tl_set:Nn \l_tmpa_tl { { #2 } }
9645     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9646     { \@@_for_a_block:nnnnn ##1 }
9647     \@@_all_the_cells:
9648     \group_end:
9649   }
9650   { \@@_error:n { TikzEveryCell~without~tikz } }
9651 }
9652
9653
9654 \cs_new_protected:Nn \@@_all_the_cells:
9655 {
9656   \int_step_inline:nn \c@iRow
9657   {
9658     \int_step_inline:nn \c@jCol
9659     {
9660       \cs_if_exist:cF { cell - ##1 - #####1 }
9661       {
9662         \clist_if_in:Nef \l_@@_corners_cells_clist
9663         { ##1 - #####1 }
9664         {
9665           \bool_set_false:N \l_tmpa_bool
9666           \cs_if_exist:cTF
9667           { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
9668           {
9669             \bool_if:NF \l_@@_empty_bool
9670             { \bool_set_true:N \l_tmpa_bool }
9671           }
9672           {
9673             \bool_if:NF \l_@@_not_empty_bool
9674             { \bool_set_true:N \l_tmpa_bool }
9675           }
9676           \bool_if:NT \l_tmpa_bool
9677           {
9678             \@@_block_tikz:nnnnn
9679             \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
9680           }
9681         }
9682       }
9683     }
9684   }
9685 }
9686
9687 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9688 {
9689   \bool_if:NF \l_@@_empty_bool
9690   {
9691     \@@_block_tikz:nnnnn
9692     \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9693   }
9694   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9695 }
9696
9697 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn

```

```

9698 {
9699   \int_step_inline:nnn { #1 } { #3 }
9700   {
9701     \int_step_inline:nnn { #2 } { #4 }
9702     { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9703   }
9704 }

```

37 The command \ShowCellNames

```

9705 \NewDocumentCommand \@@_ShowCellNames { }
9706 {
9707   \bool_if:NT \l_@@_in_code_after_bool
9708   {
9709     \pgfpicture
9710     \pgfrememberpicturepositiononpagetrue
9711     \pgf@relevantforpicturesizefalse
9712     \pgfpathrectanglecorners
9713     { \@@_qpoint:n { 1 } }
9714     {
9715       \@@_qpoint:n
9716       { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
9717     }
9718     \pgfsetfillopacity { 0.75 }
9719     \pgfsetfillcolor { white }
9720     \pgfusepathqfill
9721     \endpgfpicture
9722   }
9723   \dim_gzero_new:N \g_@@_tmpc_dim
9724   \dim_gzero_new:N \g_@@_tmpd_dim
9725   \dim_gzero_new:N \g_@@_tmpe_dim
9726   \int_step_inline:nn { \c@iRow }
9727   {
9728     \bool_if:NTF \l_@@_in_code_after_bool
9729     {
9730       \pgfpicture
9731       \pgfrememberpicturepositiononpagetrue
9732       \pgf@relevantforpicturesizefalse
9733     }
9734     { \begin { pgfpicture } }
9735     \@@_qpoint:n { row - ##1 }
9736     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9737     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9738     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9739     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9740     \bool_if:NTF \l_@@_in_code_after_bool
9741     { \endpgfpicture }
9742     { \end { pgfpicture } }
9743     \int_step_inline:nn { \c@jCol }
9744     {
9745       \hbox_set:Nn \l_tmpa_box
9746       {
9747         \normalfont \Large \sffamily \bfseries
9748         \bool_if:NTF \l_@@_in_code_after_bool
9749         { \color { red } }
9750         { \color { red ! 50 } }
9751         ##1 - #####1
9752       }
9753       \bool_if:NTF \l_@@_in_code_after_bool
9754       {
9755         \pgfpicture
9756         \pgfrememberpicturepositiononpagetrue

```

```

9757         \pgf@relevantforpicturesizefalse
9758     }
9759     { \begin { pgfpicture } }
9760     \@@_qpoint:n { col - #####1 }
9761     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9762     \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
9763     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9764     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9765     \bool_if:NTF \l_@@_in_code_after_bool
9766     { \endpgfpicture }
9767     { \end { pgfpicture } }
9768     \fp_set:Nn \l_tmpa_fp
9769     {
9770         \fp_min:nn
9771         {
9772             \fp_min:nn
9773             { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9774             { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9775         }
9776         { 1.0 }
9777     }
9778     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9779     \pgfpicture
9780     \pgfrememberpicturepositiononpagetrue
9781     \pgf@relevantforpicturesizefalse
9782     \pgftransformshift
9783     {
9784         \pgfpoint
9785         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9786         { \dim_use:N \g_tmpa_dim }
9787     }
9788     \pgfnode
9789     { rectangle }
9790     { center }
9791     { \box_use:N \l_tmpa_box }
9792     { }
9793     { }
9794     \endpgfpicture
9795 }
9796 }
9797 }

```

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.

```

9798 \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.

```

9799 \bool_new:N \g_@@_footnote_bool
9800 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9801 {
9802     You-have-used-the-key~' \l_keys_key_str '~when-loading-nicematrix~
9803     but-that-key-is-unknown. \l
9804     It-will-be-ignored. \l

```

```

9805   For~a~list~of~the~available~keys,~type~H~<return>.
9806 }
9807 {
9808   The~available~keys~are~(in~alphabetic~order):~
9809   footnote,~
9810   footnotehyper,~
9811   messages-for-Overleaf,~
9812   renew-dots~and~
9813   renew-matrix.
9814 }
9815 \keys_define:nn { nicematrix }
9816 {
9817   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9818   renew-dots .value_forbidden:n = true ,
9819   renew-matrix .code:n = \@@_renew_matrix: ,
9820   renew-matrix .value_forbidden:n = true ,
9821   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9822   footnote .bool_set:N = \g_@@_footnote_bool ,
9823   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9824   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9825 }
9826 \ProcessKeyOptions

9827 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9828 {
9829   You~can't~use~the~option~'footnote'~because~the~package~
9830   footnotehyper~has~already~been~loaded.~
9831   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9832   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9833   of~the~package~footnotehyper.\\
9834   The~package~footnote~won't~be~loaded.
9835 }
9836 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9837 {
9838   You~can't~use~the~option~'footnotehyper'~because~the~package~
9839   footnote~has~already~been~loaded.~
9840   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9841   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9842   of~the~package~footnote.\\
9843   The~package~footnotehyper~won't~be~loaded.
9844 }

9845 \bool_if:NT \g_@@_footnote_bool
9846 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9847 \IfClassLoadedTF { beamer }
9848 { \bool_set_false:N \g_@@_footnote_bool }
9849 {
9850   \IfPackageLoadedTF { footnotehyper }
9851   { \@@_error:n { footnote~with~footnotehyper~package } }
9852   { \usepackage { footnote } }
9853 }
9854 }
9855 \bool_if:NT \g_@@_footnotehyper_bool
9856 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9857 \IfClassLoadedTF { beamer }

```



```

9858     { \bool_set_false:N \g_@@_footnote_bool }
9859     {
9860       \IfPackageLoadedTF { footnote }
9861       { \@@_error:n { footnotehyper~with~footnote~package } }
9862       { \usepackage { footnotehyper } }
9863     }
9864     \bool_set_true:N \g_@@_footnote_bool
9865   }

```

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.

```

9866 \bool_new:N \l_@@_underscore_loaded_bool
9867 \IfPackageLoadedT { underscore }
9868 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9869 \hook_gput_code:nnn { begindocument } { . }
9870 {
9871   \bool_if:NF \l_@@_underscore_loaded_bool
9872   {
9873     \IfPackageLoadedT { underscore }
9874     { \@@_error:n { underscore~after~nicematrix } }
9875   }
9876 }

```

40 Error messages of the package

When there is a unknown key, we try a “normal form” of the key and, when that normal form exists, we add that information in the error message.

The normal form is the lower case form of the key, with all the spaces replaced by hyphens (there is never spaces in the keys of `nicematrix`).

#1 is a clist of names of sets of keys and **#2** is the error message to send.

```

9877 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
9878 {
9879   \str_set_eq:NN \l_tmpa_str \l_keys_key_str
9880   \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
9881   \str_set:Nx \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
9882   \bool_set_false:N \l_tmpa_bool
9883   \clist_map_inline:nn { #1 }
9884   {
9885     \keys_if_exist:neT { ##1 } { \l_tmpa_str }
9886     {
9887       \@@_error:n { key~with~normal~form~exists }
9888       \bool_set_true:N \l_tmpa_bool
9889       \clist_map_break:
9890     }
9891   }
9892   \bool_if:NF \l_tmpa_bool { \@@_error:n { #2 } }
9893 }

```

```

9894 \@@_msg_new:nn { key~with~normal~form~exists }
9895 {
9896   The~key~'\l_keys_key_str'~does~not~exists.\\
9897   It~will~be~ignored.\\
9898   Maybe~you~want~to~use~the~key~'\l_tmpa_str'.
9899 }
9900 \str_const:Ne \c_@@_available_keys_str
9901 {
9902   \bool_if:nT { ! \g_@@_messages_for_Overleaf_bool }
9903   { For~a~list~of~the~available~keys,~type~H~<return>. }
9904 }
9905 \seq_new:N \g_@@_types_of_matrix_seq
9906 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9907 {
9908   NiceMatrix ,
9909   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9910 }
9911 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9912 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_err_too_many_cols:` is triggered. This command raises an error but also tries to give the best information to the user in the error message. The command `\seq_if_in:NoF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

9913 \cs_new_protected:Npn \@@_err_too_many_cols:
9914 {
9915   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9916   { \@@_fatal:nn { too~many~cols~for~array } }
9917   \int_compare:nNnT { \l_@@_last_col_int } = { -2 }
9918   { \@@_fatal:n { too~many~cols~for~matrix } }
9919   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
9920   { \@@_fatal:n { too~many~cols~for~matrix } }
9921   \bool_if:NF \l_@@_last_col_without_value_bool
9922   { \@@_fatal:n { too~many~cols~for~matrix~with~last~col } }
9923 }

```

The following command must *not* be protected since it's used in an error message.

```

9924 \cs_new:Npn \@@_message_hdotsfor:
9925 {
9926   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9927   { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
9928     \token_to_str:N \Hbrace \ is~incorrect. }
9929 }

9930 \cs_new_protected:Npn \@@_Hline_in_cell:
9931 { \@@_fatal:n { Misuse~of~Hline } }

9932 \@@_msg_new:nn { Misuse~of~Hline }
9933 {
9934   Misuse~of~Hline. \\
9935   Error~in~your~row~ \int_eval:n { \c@iRow }. \\
9936   \token_to_str:N \Hline\ must~be~used~only~at~the~beginning~of~a~row.\\
9937   That~error~is~fatal.
9938 }

9939 \@@_msg_new:nn { hvlines,~rounded~corners~and~corners }
9940 {
9941   Incompatible~options.\\
9942   You~should~not~use~'hvlines',~'rounded~corners'~and~'corners'~at~the~same~time.\\
9943   The~output~will~not~be~reliable.
9944 }

```

```

9945 \@@_msg_new:nn { Body-alone }
9946 {
9947   \token_to_str:N \Body\ alone. \\
9948   You-have-used-\token_to_str:N \Body\ without-\token_to_str:N \CodeBefore.\\
9949   That~error~is~fatal.
9950 }
9951 \@@_msg_new:nn { cellcolor-in-Block }
9952 {
9953   Bad-use-of-\token_to_str:N \cellcolor \\
9954   You-can't-use-\token_to_str:N \cellcolor\ in-\token_to_str:N \Block\
9955   (except~in~a~sub~block).\\
9956   That~command~will~be~ignored.
9957 }
9958 \@@_msg_new:nn { rowcolor-in-Block }
9959 {
9960   Bad-use-of-\token_to_str:N \rowcolor \\
9961   You-can't-use-\token_to_str:N \rowcolor\ in-\token_to_str:N \Block.\\
9962   That~command~will~be~ignored.
9963 }
9964 \@@_msg_new:nn { key-color-inside }
9965 {
9966   Deleted~key.\\
9967   The~key~'color-inside'~(and~its~alias~'colortbl-like')~has~been~deleted~in
9968   ~'nicematrix'~and~must~not~be~used.\\
9969   This~error~is~fatal.
9970 }
9971 \@@_msg_new:nn { invalid-weight }
9972 {
9973   Unknown~key.\\
9974   The~key~' \l_keys_key_str '~-of~your~column~X~is~unknown~and~will~be~ignored.
9975 }
9976 \@@_msg_new:nn { last-col-not-used }
9977 {
9978   Column~not~used.\\
9979   The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
9980   in~your~\@@_full_name_env: .~
9981   However,~you~can~go~on.
9982 }
9983 \@@_msg_new:nn { too-many-cols-for-matrix-with-last-col }
9984 {
9985   Too-many-columns.\\
9986   In~the~row~ \int_eval:n { \c@iRow },~
9987   you~try~to~use~more~columns~
9988   than~allowed~by~your~ \@@_full_name_env: .
9989   \@@_message_hdotsfor: \
9990   The~maximal~number~of~columns~is~ \int_eval:n { \l_@@_last_col_int - 1 }~
9991   (plus~the~exterior~columns).~This~error~is~fatal.
9992 }
9993 \@@_msg_new:nn { too-many-cols-for-matrix }
9994 {
9995   Too-many-columns.\\
9996   In~the~row~ \int_eval:n { \c@iRow },~
9997   you~try~to~use~more~columns~than~allowed~by~your~ \@@_full_name_env: .
9998   \@@_message_hdotsfor: \
9999   Recall~that~the~maximal~number~of~columns~for~a~matrix~
10000   (excepted~the~potential~exterior~columns)~is~fixed~by~the~
10001   LaTeX~counter~'MaxMatrixCols'.~
10002   Its~current~value~is~ \int_use:N \c@MaxMatrixCols \
10003   (use~ \token_to_str:N \setcounter \ to~change~that~value).~
10004   This~error~is~fatal.
10005 }

```

```

10006 \@@_msg_new:nn { too-many-cols-for-array }
10007 {
10008   Too-many-columns.\\
10009   In-the-row~ \int_eval:n { \c@iRow } ,~
10010   ~you-try-to-use-more-columns-than-allowed-by-your~
10011   \@@_full_name_env: . \@@_message_hdotsfor: \ The-maximal-number-of-columns-is~
10012   \int_use:N \g_@@_static_num_of_col_int \
10013   \bool_if:nT
10014     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
10015     { (plus-the-exterior-ones)~}
10016   since-the-preamble-is~' \g_@@_user_preamble_tl '.\\
10017   This-error-is-fatal.
10018 }
10019 \@@_msg_new:nn { columns-not-used }
10020 {
10021   Columns-not-used.\\
10022   The-preamble-of-your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '~
10023   It-announces~ \int_use:N \g_@@_static_num_of_col_int \
10024   columns-but-you-only-used~ \int_use:N \c@jCol .\\
10025   The-columns-you-did-not-used-won't-be-created.\\
10026   You-won't-have-similar-warning-till-the-end-of-the-document.
10027 }
10028 \@@_msg_new:nn { empty-preamble }
10029 {
10030   Empty-preamble.\\
10031   The-preamble-of-your~ \@@_full_name_env: \ is-empty.\\
10032   This-error-is-fatal.
10033 }
10034 \@@_msg_new:nn { in-first-col }
10035 {
10036   Erroneous-use.\\
10037   You-can't-use-the-command-#1 in-the-first-column-(number-0)-of-the-array.\\
10038   That-command-will-be-ignored.
10039 }
10040 \@@_msg_new:nn { in-last-col }
10041 {
10042   Erroneous-use.\\
10043   You-can't-use-the-command-#1 in-the-last-column-(exterior)-of-the-array.\\
10044   That-command-will-be-ignored.
10045 }
10046 \@@_msg_new:nn { in-first-row }
10047 {
10048   Erroneous-use.\\
10049   You-can't-use-the-command-#1 in-the-first-row-(number-0)-of-the-array.\\
10050   That-command-will-be-ignored.
10051 }
10052 \@@_msg_new:nn { in-last-row }
10053 {
10054   Erroneous-use.\\
10055   You-can't-use-the-command-#1 in-the-last-row-(exterior)-of-the-array.\\
10056   That-command-will-be-ignored.
10057 }
10058 \@@_msg_new:nn { TopRule-without-booktabs }
10059 {
10060   Erroneous-use.\\
10061   You-can't-use-the-command~ #1 because-'booktabs'~is-not-loaded.\\
10062   That-command-will-be-ignored.
10063 }
10064 \@@_msg_new:nn { TopRule-without-tikz }
10065 {

```

```

10066     Erroneous~use.\\
10067     You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
10068     That~command~will~be~ignored.
10069 }

10070 \@@_msg_new:nn { caption~outside~float }
10071 {
10072     Key~caption~forbidden.\\
10073     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
10074     environment~(such~as~\{table\}).~This~key~will~be~ignored.
10075 }

10076 \@@_msg_new:nn { short~caption~without~caption }
10077 {
10078     You~should~not~use~the~key~'short~caption'~without~'caption'.~
10079     However,~your~'short~caption'~will~be~used~as~'caption'.
10080 }

10081 \@@_msg_new:nn { double~closing~delimiter }
10082 {
10083     Double~delimiter.\\
10084     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
10085     delimiter.~This~delimiter~will~be~ignored.
10086 }

10087 \@@_msg_new:nn { delimiter~after~opening }
10088 {
10089     Double~delimiter.\\
10090     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
10091     delimiter.~That~delimiter~will~be~ignored.
10092 }

10093 \@@_msg_new:nn { bad~option~for~line~style }
10094 {
10095     Bad~line~style.\\
10096     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
10097     is~'standard'.~That~key~will~be~ignored.
10098 }

10099 \@@_msg_new:nn { corners~with~no~cell~nodes }
10100 {
10101     Incompatible~keys.\\
10102     You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
10103     is~in~force.\\
10104     If~you~go~on,~that~key~will~be~ignored.
10105 }

10106 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
10107 {
10108     Incompatible~keys.\\
10109     You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
10110     is~in~force.\\
10111     If~you~go~on,~those~extra~nodes~won't~be~created.
10112 }

10113 \@@_msg_new:nn { Identical~notes~in~caption }
10114 {
10115     Identical~tabular~notes.\\
10116     You~can't~put~several~notes~with~the~same~content~in~
10117     \token_to_str:N \caption \ (but~you~can~in~the~main~tabular).\\
10118     If~you~go~on,~the~output~will~probably~be~erroneous.
10119 }

10120 \@@_msg_new:nn { tabularnote~below~the~tabular }
10121 {
10122     \token_to_str:N \tabularnote \ forbidden\\
10123     You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
10124     of~your~tabular~because~the~caption~will~be~composed~below~
10125     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~

```

```

10126     key~'caption-above'~in~ \token_to_str:N \NiceMatrixOptions .\\
10127     Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
10128     no~similar~error~will~raised~in~this~document.
10129 }
10130 \@@_msg_new:nn { Unknown~key~for~rules }
10131 {
10132     Unknown~key.\\
10133     There~is~only~two~keys~available~here:~width~and~color.\\
10134     Your~key~' \l_keys_key_str '~will~be~ignored.
10135 }
10136 \@@_msg_new:nn { Unknown~key~for~Hbrace }
10137 {
10138     Unknown~key.\\
10139     You~have~used~the~key~' \l_keys_key_str '~but~the~only~
10140     keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
10141     and~ \token_to_str:N \Vbrace \ are:~'brace-shift',~'color',~
10142     'horizontal-label(s)',~'shorten'~'shorten-end'~
10143     and~'shorten-start'.\\
10144     That~error~is~fatal.
10145 }
10146 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
10147 {
10148     Unknown~key.\\
10149     There~is~only~two~keys~available~here:~
10150     'empty'~and~'not-empty'.\\
10151     Your~key~' \l_keys_key_str '~will~be~ignored.
10152 }
10153 \@@_msg_new:nn { Unknown~key~for~rotate }
10154 {
10155     Unknown~key.\\
10156     The~only~key~available~here~is~'c'.\\
10157     Your~key~' \l_keys_key_str '~will~be~ignored.
10158 }
10159 \@@_msg_new:nnn { Unknown~key~for~custom-line }
10160 {
10161     Unknown~key.\\
10162     The~key~' \l_keys_key_str '~is~unknown~in~a~'custom-line'.~
10163     It~you~go~on,~you~will~probably~have~other~errors. \\
10164     \c_@@_available_keys_str
10165 }
10166 {
10167     The~available~keys~are~(in~alphabetic~order):~
10168     ccommand,~
10169     color,~
10170     command,~
10171     dotted,~
10172     letter,~
10173     multiplicity,~
10174     sep-color,~
10175     tikz,~and~total-width.
10176 }
10177 \@@_msg_new:nnn { Unknown~key~for~xdots }
10178 {
10179     Unknown~key.\\
10180     The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
10181     \c_@@_available_keys_str
10182 }
10183 {
10184     The~available~keys~are~(in~alphabetic~order):~
10185     'color',~
10186     'horizontal(s)-labels',~

```

```

10187     'inter',~
10188     'line-style',~
10189     'nullify',~
10190     'radius',~
10191     'shorten',~
10192     'shorten-end'~and~'shorten-start'.
10193 }
10194 \@@_msg_new:nn { Unknown~key~for~rowcolors }
10195 {
10196     Unknown~key.\\
10197     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
10198     (and~you~try~to~use~' \l_keys_key_str ')\
10199     That~key~will~be~ignored.
10200 }
10201 \@@_msg_new:nn { label~without~caption }
10202 {
10203     You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
10204     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
10205 }
10206 \@@_msg_new:nn { W~warning }
10207 {
10208     Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
10209     (row~ \int_use:N \c@iRow ).
10210 }
10211 \@@_msg_new:nn { Construct~too~large }
10212 {
10213     Construct~too~large.\\
10214     Your~command~ \token_to_str:N #1
10215     can't~be~drawn~because~your~matrix~is~too~small.\\
10216     That~command~will~be~ignored.
10217 }
10218 \@@_msg_new:nn { underscore~after~nicematrix }
10219 {
10220     Problem~with~'underscore'.\\
10221     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
10222     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
10223     ' \token_to_str:N \Cdots \token_to_str:N _
10224     \{ n \token_to_str:N \text \{ ~times \} \}'.
10225 }
10226 \@@_msg_new:nn { ampersand~in~light-syntax }
10227 {
10228     Ampersand~forbidden.\\
10229     You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
10230     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
10231 }
10232 \@@_msg_new:nn { double~backslash~in~light-syntax }
10233 {
10234     Double~backslash~forbidden.\\
10235     You~can't~use~ \token_to_str:N \
10236     ~to~separate~rows~because~the~key~'light-syntax'~
10237     is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
10238     (set~by~the~key~'end-of-row').~This~error~is~fatal.
10239 }
10240 \@@_msg_new:nn { hlines~with~color }
10241 {
10242     Incompatible~keys.\\
10243     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
10244     \token_to_str:N \Block \ when~the~key~'color'~or~'draw'~is~used.\\
10245     However,~you~can~put~several~commands~ \token_to_str:N \Block.\\
10246     Your~key~will~be~discarded.
10247 }

```

```

10248 \@@_msg_new:nn { bad-value-for-baseline }
10249 {
10250   Bad-value-for-baseline.\
10251   The-value-given-to-'baseline'~( \int_use:N \l_tmpa_int )~is-not~
10252   valid.~The-value-must-be-between-\int_use:N \l_@@_first_row_int\ and~
10253   \int_use:N \g_@@_row_total_int \ or-equal-to-'t',~'c'~or~'b'~or-of~
10254   the-form-'line-i'.\
10255   A-value-of-1~will-be-used.
10256 }
10257 \@@_msg_new:nn { bad-value-for-baseline-line }
10258 {
10259   Bad-value-for-baseline-with-line.\
10260   The-value-given-to-'baseline'~( \int_use:N \l_tmpa_int )~is-not~
10261   valid.~The-number-of-the-line-must-be-between-1~and~
10262   \int_eval:n { \c@iRow + 1 } \
10263   A-value-of-'line-1'~will-be-used.
10264 }
10265 \@@_msg_new:nn { detection-of-empty-cells }
10266 {
10267   Problem-with-'not-empty'\
10268   For-technical-reasons,~you-must-activate~
10269   'create-cell-nodes'~in~ \token_to_str:N \CodeBefore \
10270   in-order-to-use-the-key~' \l_keys_key_str '.\
10271   That-key-will-be-ignored.
10272 }
10273 \@@_msg_new:nn { siunitx-not-loaded }
10274 {
10275   siunitx-not-loaded\
10276   You-can't-use-the-columns-'S'~because-'siunitx'~is-not-loaded.\
10277   That-error-is-fatal.
10278 }
10279 \@@_msg_new:nn { Invalid-name }
10280 {
10281   Invalid-name.\
10282   You-can't-give-the-name-' \l_keys_value_tl '~to-a~ \token_to_str:N
10283   \SubMatrix \ of-your~ \@@_full_name_env: .\
10284   A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\
10285   This-key-will-be-ignored.
10286 }
10287 \@@_msg_new:nn { Hbrace-not-allowed }
10288 {
10289   Command-not-allowed.\
10290   You-can't-use-the-command~ \token_to_str:N #1
10291   because-you-have-not-loaded~
10292   \IfPackageLoadedTF { tikz }
10293   { the-TikZ-library~'decorations.pathreplacing'~Use~ }
10294   { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
10295   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \
10296   That-command-will-be-ignored.
10297 }
10298 \@@_msg_new:nn { Vbrace-not-allowed }
10299 {
10300   Command-not-allowed.\
10301   You-can't-use-the-command~ \token_to_str:N \Vbrace \
10302   because-you-have-not-loaded-TikZ~
10303   and-the-TikZ-library~'decorations.pathreplacing'.\
10304   Use: ~\token_to_str:N \usepackage \{tikz\}~
10305   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \
10306   That-command-will-be-ignored.
10307 }
10308 \@@_msg_new:nn { Wrong-line-in-SubMatrix }

```



```

10309 {
10310     Wrong~line.\\
10311     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
10312     \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
10313     number~is~not~valid.~It~will~be~ignored.
10314 }
10315 \@@_msg_new:nn { Impossible~delimiter }
10316 {
10317     Impossible~delimiter.\\
10318     It's~impossible~to~draw~the~#1~delimiter~of~your~
10319     \token_to_str:N \SubMatrix \ because~all~the~cells~are~empty~
10320     in~that~column.
10321     \bool_if:NT \l_@@_submatrix_slim_bool
10322     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
10323     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
10324 }
10325 \@@_msg_new:nnn { width~without~X~columns }
10326 {
10327     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
10328     the~preamble~(' \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .\\
10329     That~key~will~be~ignored.
10330 }
10331 {
10332     This~message~is~the~message~'width~without~X~columns'~
10333     of~the~module~'nicematrix'.~
10334     The~experimented~users~can~disable~that~message~with~
10335     \token_to_str:N \msg_redirect_name:nnn .\\
10336 }
10337
10338 \@@_msg_new:nn { key~multiplicity~with~dotted }
10339 {
10340     Incompatible~keys. \\
10341     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
10342     in~a~'custom~line'.~They~are~incompatible. \\
10343     The~key~'multiplicity'~will~be~discarded.
10344 }
10345 \@@_msg_new:nn { empty~environment }
10346 {
10347     Empty~environment.\\
10348     Your~ \@@_full_name_env: \ is~empty.~This~error~is~fatal.
10349 }
10350 \@@_msg_new:nn { No~letter~and~no~command }
10351 {
10352     Erroneous~use.\\
10353     Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
10354     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10355     '~'ccommand'~(to~draw~horizontal~rules).\\
10356     However,~you~can~go~on.
10357 }
10358 \@@_msg_new:nn { Forbidden~letter }
10359 {
10360     Forbidden~letter.\\
10361     You~can't~use~the~letter~'#1'~for~a~customized~line.~
10362     It~will~be~ignored.\\
10363     The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10364 }
10365 \@@_msg_new:nn { Several~letters }
10366 {
10367     Wrong~name.\\
10368     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10369     have~used~' \l_@@_letter_str ').\\

```

```

10370     It~will~be~ignored.
10371 }

10372 \@@_msg_new:nn { Delimiter-with-small }
10373 {
10374     Delimiter~forbidden.\\
10375     You~can't~put~a~delimiter~in~the~preamble~of~your~
10376     \@@_full_name_env: \
10377     because~the~key~'small'~is~in~force.\\
10378     This~error~is~fatal.
10379 }

10380 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10381 {
10382     Unknown~cell.\\
10383     Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10384     the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10385     can't~be~executed~because~a~cell~doesn't~exist.\\
10386     This~command~ \token_to_str:N \line \ will~be~ignored.
10387 }

10388 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10389 {
10390     Duplicate~name.\\
10391     The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10392     in~this~ \@@_full_name_env: .\\
10393     This~key~will~be~ignored.\\
10394     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10395     { For~a~list~of~the~names~already~used,~type~H<return>. }
10396 }
10397 {
10398     The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
10399     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10400 }

10401 \@@_msg_new:nn { r~or~l~with~preamble }
10402 {
10403     Erroneous~use.\\
10404     You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
10405     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10406     your~ \@@_full_name_env: .\\
10407     This~key~will~be~ignored.
10408 }

10409 \@@_msg_new:nn { Hdotsfor~in~col-0 }
10410 {
10411     Erroneous~use.\\
10412     You~can't~use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\
10413     in~an~exterior~column~of~
10414     the~array.~This~error~is~fatal.
10415 }

10416 \@@_msg_new:nn { bad~corner }
10417 {
10418     Bad~corner.\\
10419     #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10420     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
10421     This~specification~of~corner~will~be~ignored.
10422 }

10423 \@@_msg_new:nn { bad~border }
10424 {
10425     Bad~border.\\
10426     \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
10427     (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
10428     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10429     also~use~the~key~'tikz'
10430     \IfPackageLoadedF { tikz }

```

```

10431     { ~if-you-load-the-LaTeX-package~'tikz' } ).\\
10432     This~specification-of~border~will~be~ignored.
10433 }

10434 \@@_msg_new:nn { TikzEveryCell~without~tikz }
10435 {
10436     TikZ~not~loaded.\\
10437     You~can't~use~ \token_to_str:N \TikzEveryCell \
10438     because~you~have~not~loaded~tikz.~
10439     This~command~will~be~ignored.
10440 }

10441 \@@_msg_new:nn { tikz~key~without~tikz }
10442 {
10443     TikZ~not~loaded.\\
10444     You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N
10445     \Block '~because~you~have~not~loaded~tikz.~
10446     This~key~will~be~ignored.
10447 }

10448 \@@_msg_new:nn { Bad~argument~for~Block }
10449 {
10450     Bad~argument.\\
10451     The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
10452     be~of~the~form~'i-j'~(or~completely~empty)~and~you~have~used:~
10453     '#1'. \\
10454     If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono~cell~(as~if~
10455     the~argument~was~empty).
10456 }

10457 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
10458 {
10459     Erroneous~use.\\
10460     In~the~ \@@_full_name_env: ,~you~must~use~the~key~
10461     'last~col'~without~value.\\
10462     However,~you~can~go~on~for~this~time~
10463     (the~value~' \l_keys_value_tl '~will~be~ignored).
10464 }

10465 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
10466 {
10467     Erroneous~use. \\
10468     In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
10469     'last~col'~without~value. \\
10470     However,~you~can~go~on~for~this~time~
10471     (the~value~' \l_keys_value_tl '~will~be~ignored).
10472 }

10473 \@@_msg_new:nn { Block~too~large~1 }
10474 {
10475     Block~too~large. \\
10476     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
10477     too~small~for~that~block. \\
10478     This~block~and~maybe~others~will~be~ignored.
10479 }

10480 \@@_msg_new:nn { Block~too~large~2 }
10481 {
10482     Block~too~large. \\
10483     The~preamble~of~your~ \@@_full_name_env: \ announces~ \int_use:N
10484     \g_@@_static_num_of_col_int \
10485     columns~but~you~use~only~ \int_use:N \c@jCol \ and~that's~why~a~block~
10486     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
10487     (&)~at~the~end~of~the~first~row~of~your~ \@@_full_name_env: . \\
10488     This~block~and~maybe~others~will~be~ignored.
10489 }

10490 \@@_msg_new:nn { unknown~column~type }

```

```

10491 {
10492   Bad-column-type. \\
10493   The-column-type~'#1'~in~your~ \@@_full_name_env: \
10494   is~unknown. \\
10495   This~error~is~fatal.
10496 }
10497 \@@_msg_new:nn { unknown~column-type-multicolumn }
10498 {
10499   Bad-column-type. \\
10500   The-column-type~'#1'~in~the~command~\token_to_str:N \multicolumn \
10501   ~of~your~ \@@_full_name_env: \
10502   is~unknown. \\
10503   This~error~is~fatal.
10504 }
10505 \@@_msg_new:nn { unknown~column-type-S }
10506 {
10507   Bad-column-type. \\
10508   The-column-type~'S'~in~your~ \@@_full_name_env: \ is~unknown. \\
10509   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10510   load~that~package. \\
10511   This~error~is~fatal.
10512 }
10513 \@@_msg_new:nn { unknown~column-type-S~multicolumn }
10514 {
10515   Bad-column-type. \\
10516   The-column-type~'S'~in~the~command~\token_to_str:N \multicolumn \
10517   of~your~ \@@_full_name_env: \ is~unknown. \\
10518   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10519   load~that~package. \\
10520   This~error~is~fatal.
10521 }
10522 \@@_msg_new:nn { tabularnote~forbidden }
10523 {
10524   Forbidden~command. \\
10525   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10526   ~here.~This~command~is~available~only~in~
10527   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
10528   the~argument~of~a~command~\token_to_str:N \caption \ included~
10529   in~an~environment~\{table\}. \\
10530   This~command~will~be~ignored.
10531 }
10532 \@@_msg_new:nn { borders~forbidden }
10533 {
10534   Forbidden~key.\\
10535   You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \
10536   because~the~option~'rounded~corners'~
10537   is~in~force~with~a~non~zero~value.\\
10538   This~key~will~be~ignored.
10539 }
10540 \@@_msg_new:nn { bottomrule~without~booktabs }
10541 {
10542   booktabs~not~loaded.\\
10543   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
10544   loaded~'booktabs'.\\
10545   This~key~will~be~ignored.
10546 }
10547 \@@_msg_new:nn { enumitem~not~loaded }
10548 {
10549   enumitem~not~loaded. \\
10550   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10551   ~because~you~haven't~loaded~'enumitem'. \\

```

```

10552 All-the-commands~ \token_to_str:N \tabularnote \ will-be-
10553 ignored-in-the-document.
10554 }
10555 \@@_msg_new:nn { tikz-without-tikz }
10556 {
10557 Tikz-not-loaded. \\
10558 You-can't-use-the-key~'tikz'~here~because~Tikz-is-not~
10559 loaded.~If-you-go-on,~that-key~will-be-ignored.
10560 }
10561 \@@_msg_new:nn { tikz-in-custom-line-without-tikz }
10562 {
10563 Tikz-not-loaded. \\
10564 You-have-used-the-key~'tikz'~in-the-definition-of~a~
10565 customized~line~(with~'custom-line')~but~tikz-is-not-loaded.~
10566 You-can-go-on-but-you-will-have-another-error-if-you-actually~
10567 use~that~custom~line.
10568 }
10569 \@@_msg_new:nn { tikz-in-borders-without-tikz }
10570 {
10571 Tikz-not-loaded. \\
10572 You-have-used-the-key~'tikz'~in~a~key~'borders'~(of~a~
10573 command~' \token_to_str:N \Block ')~but~tikz-is-not-loaded.~
10574 That~key~will-be-ignored.
10575 }
10576 \@@_msg_new:nn { color-in-custom-line-with-tikz }
10577 {
10578 Erroneous-use.\\
10579 In~a~'custom-line',~you-have-used-both~'tikz'~and~'color',~
10580 which-is-forbidden~(you-should-use~'color'~inside~the-key~'tikz').~
10581 The-key~'color'~will-be-discarded.
10582 }
10583 \@@_msg_new:nn { Wrong-last-row }
10584 {
10585 Wrong-number.\\
10586 You-have-used~'last-row= \int_use:N \l_@@_last_row_int '~but-your~
10587 \@@_full_name_env: \ seems-to-have~ \int_use:N \c@iRow \ rows.~
10588 If-you-go-on,~the-value-of~ \int_use:N \c@iRow \ will-be-used-for~
10589 last-row-but-you-should-correct-your-code.~You-can-avoid-this~
10590 problem-by-using~'last-row'~without~value~(more-compilations~
10591 might-be-necessary).
10592 }
10593 \@@_msg_new:nn { Yet-in-env }
10594 {
10595 Nested-environments.\\
10596 Environments-of~nicematrix~can't-be-nested.\\
10597 This-error-is-fatal.
10598 }
10599 \@@_msg_new:nn { Outside-math-mode }
10600 {
10601 Outside-math-mode.\\
10602 The~\@@_full_name_env: \ can-be-used-only-in-math-mode~
10603 (and-not-in~ \token_to_str:N \vcenter ).\\
10604 This-error-is-fatal.
10605 }
10606 \@@_msg_new:nn { One-letter-allowed }
10607 {
10608 Bad-name.\\
10609 The-value-of-key~' \l_keys_key_str '~must-be-of-length~1~and~
10610 you-have-used~' \l_keys_value_tl '~.\\
10611 It-will-be-ignored.
10612 }

```

```

10613 \@@_msg_new:nn { TabularNote~in~CodeAfter }
10614 {
10615   Environment~\{TabularNote\}~forbidden.\\
10616   You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
10617   but~*before*~the~ \token_to_str:N \CodeAfter . \\
10618   This~environment~\{TabularNote\}~will~be~ignored.
10619 }
10620 \@@_msg_new:nn { varwidth~not~loaded }
10621 {
10622   varwidth~not~loaded.\\
10623   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
10624   loaded.\\
10625   Your~column~will~behave~like~'p'.
10626 }
10627 \@@_msg_new:nn { varwidth~not~loaded~in~X }
10628 {
10629   varwidth~not~loaded.\\
10630   You~can't~use~the~key~'V'~in~your~column~'X'~
10631   because~'varwidth'~is~not~loaded.\\
10632   It~will~be~ignored. \\
10633 }
10634 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
10635 {
10636   Unknown~key.\\
10637   Your~key~' \l_keys_key_str '~is~unknown~for~a~rule.\\
10638   \c_@@_available_keys_str
10639 }
10640 {
10641   The~available~keys~are~(in~alphabetic~order):~
10642   color,~
10643   dotted,~
10644   multiplicity,~
10645   sep~color,~
10646   tikz,~and~total~width.
10647 }
10648
10649 \@@_msg_new:nnn { Unknown~key~for~Block }
10650 {
10651   Unknown~key. \\
10652   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10653   \token_to_str:N \Block . \\
10654   It~will~be~ignored. \\
10655   \c_@@_available_keys_str
10656 }
10657 {
10658   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10659   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line~width,~name,~
10660   opacity,~rounded~corners,~r,~respect~arraystretch,~t,~T,~tikz,~transparent~
10661   and~vlines.
10662 }
10663 \@@_msg_new:nnn { Unknown~key~for~Brace }
10664 {
10665   Unknown~key.\\
10666   The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
10667   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
10668   It~will~be~ignored. \\
10669   \c_@@_available_keys_str
10670 }
10671 {
10672   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
10673   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
10674   right~shorten)~and~yshift.

```

```

10675 }
10676 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10677 {
10678   Unknown~key.\\
10679   The~key~' \l_keys_key_str '~is~unknown.\\
10680   It~will~be~ignored. \\
10681   \c_@@_available_keys_str
10682 }
10683 {
10684   The~available~keys~are~(in~alphabetic~order):~
10685   delimiters/color,~
10686   rules~(with~the~subkeys~'color'~and~'width'),~
10687   sub-matrix~(several~subkeys)~
10688   and~xdots~(several~subkeys).~
10689   The~latter~is~for~the~command~ \token_to_str:N \line .
10690 }
10691 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10692 {
10693   Unknown~key.\\
10694   The~key~' \l_keys_key_str '~is~unknown.\\
10695   It~will~be~ignored. \\
10696   \c_@@_available_keys_str
10697 }
10698 {
10699   The~available~keys~are~(in~alphabetic~order):~
10700   create-cell-nodes,~
10701   delimiters/color~and~
10702   sub-matrix~(several~subkeys).
10703 }
10704 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10705 {
10706   Unknown~key.\\
10707   The~key~' \l_keys_key_str '~is~unknown.\\
10708   That~key~will~be~ignored. \\
10709   \c_@@_available_keys_str
10710 }
10711 {
10712   The~available~keys~are~(in~alphabetic~order):~
10713   'delimiters/color',~
10714   'extra-height',~
10715   'hlines',~
10716   'hvlines',~
10717   'left-xshift',~
10718   'name',~
10719   'right-xshift',~
10720   'rules'~(with~the~subkeys~'color'~and~'width'),~
10721   'slim',~
10722   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10723   and~'right-xshift').\\
10724 }
10725 \@@_msg_new:nnn { Unknown~key~for~notes }
10726 {
10727   Unknown~key.\\
10728   The~key~' \l_keys_key_str '~is~unknown.\\
10729   That~key~will~be~ignored. \\
10730   \c_@@_available_keys_str
10731 }
10732 {
10733   The~available~keys~are~(in~alphabetic~order):~
10734   bottomrule,~
10735   code-after,~
10736   code-before,~

```

```

10737     detect-duplicates,~
10738     enumitem-keys,~
10739     enumitem-keys-para,~
10740     para,~
10741     label-in-list,~
10742     label-in-tabular~and~
10743     style.
10744 }

10745 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10746 {
10747     Unknown~key.\\
10748     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10749     \token_to_str:N \RowStyle . \\
10750     That~key~will~be~ignored. \\
10751     \c_@@_available_keys_str
10752 }
10753 {
10754     The~available~keys~are~(in~alphabetic~order):~
10755     bold,~
10756     cell-space-top-limit,~
10757     cell-space-bottom-limit,~
10758     cell-space-limits,~
10759     color,~
10760     fill~(alias:~rowcolor),~
10761     nb-rows,~
10762     opacity~and~
10763     rounded-corners.
10764 }

10765 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10766 {
10767     Unknown~key.\\
10768     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10769     \token_to_str:N \NiceMatrixOptions . \\
10770     That~key~will~be~ignored. \\
10771     \c_@@_available_keys_str
10772 }
10773 {
10774     The~available~keys~are~(in~alphabetic~order):~
10775     &~in~blocks,~
10776     allow-duplicate-names,~
10777     ampersand-in-blocks,~
10778     caption-above,~
10779     cell-space-bottom-limit,~
10780     cell-space-limits,~
10781     cell-space-top-limit,~
10782     code-for-first-col,~
10783     code-for-first-row,~
10784     code-for-last-col,~
10785     code-for-last-row,~
10786     corners,~
10787     custom-key,~
10788     create-extra-nodes,~
10789     create-medium-nodes,~
10790     create-large-nodes,~
10791     custom-line,~
10792     delimiters~(several~subkeys),~
10793     end-of-row,~
10794     first-col,~
10795     first-row,~
10796     hlines,~
10797     hvlines,~
10798     hvlines-except-borders,~
10799     last-col,~

```



```

10800     last-row,~
10801     left-margin,~
10802     light-syntax,~
10803     light-syntax-expanded,~
10804     matrix/columns-type,~
10805     no-cell-nodes,~
10806     notes~(several~subkeys),~
10807     nullify-dots,~
10808     pgf-node-code,~
10809     renew-dots,~
10810     renew-matrix,~
10811     respect-arraystretch,~
10812     rounded-corners,~
10813     right-margin,~
10814     rules~(with~the~subkeys~'color'~and~'width'),~
10815     small,~
10816     sub-matrix~(several~subkeys),~
10817     vlines,~
10818     xdots~(several~subkeys).
10819 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

10820 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10821 {
10822     Unknown~key.\\
10823     The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
10824     \{NiceArray\}. \\
10825     That~key~will~be~ignored. \\
10826     \c_@@_available_keys_str
10827 }
10828 {
10829     The~available~keys~are~(in~alphabetic~order):~
10830     &-in-blocks,~
10831     ampersand-in-blocks,~
10832     b,~
10833     baseline,~
10834     c,~
10835     cell-space-bottom-limit,~
10836     cell-space-limits,~
10837     cell-space-top-limit,~
10838     code-after,~
10839     code-for-first-col,~
10840     code-for-first-row,~
10841     code-for-last-col,~
10842     code-for-last-row,~
10843     columns-width,~
10844     corners,~
10845     create-extra-nodes,~
10846     create-medium-nodes,~
10847     create-large-nodes,~
10848     extra-left-margin,~
10849     extra-right-margin,~
10850     first-col,~
10851     first-row,~
10852     hlines,~
10853     hvlines,~
10854     hvlines-except-borders,~
10855     last-col,~
10856     last-row,~
10857     left-margin,~
10858     light-syntax,~
10859     light-syntax-expanded,~
10860     name,~

```

```

10861 no-cell-nodes,~
10862 nullify-dots,~
10863 pgf-node-code,~
10864 renew-dots,~
10865 respect-arraystretch,~
10866 right-margin,~
10867 rounded-corners,~
10868 rules~(with~the~subkeys~'color'~and~'width'),~
10869 small,~
10870 t,~
10871 vlines,~
10872 xdots/color,~
10873 xdots/shorten-start,~
10874 xdots/shorten-end,~
10875 xdots/shorten-and~
10876 xdots/line-style.
10877 }

```

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`).

```

10878 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10879 {
10880   Unknown~key.\\
10881   The~key~' \l_keys_key_str '~is~unknown~for~the~
10882   \@@_full_name_env: . \\
10883   That~key~will~be~ignored. \\
10884   \c_@@_available_keys_str
10885 }
10886 {
10887   The~available~keys~are~(in~alphabetic~order):~
10888   &-in-blocks,~
10889   ampersand-in-blocks,~
10890   b,~
10891   baseline,~
10892   c,~
10893   cell-space-bottom-limit,~
10894   cell-space-limits,~
10895   cell-space-top-limit,~
10896   code-after,~
10897   code-for-first-col,~
10898   code-for-first-row,~
10899   code-for-last-col,~
10900   code-for-last-row,~
10901   columns-type,~
10902   columns-width,~
10903   corners,~
10904   create-extra-nodes,~
10905   create-medium-nodes,~
10906   create-large-nodes,~
10907   extra-left-margin,~
10908   extra-right-margin,~
10909   first-col,~
10910   first-row,~
10911   hlines,~
10912   hvlines,~
10913   hvlines-except-borders,~
10914   l,~
10915   last-col,~
10916   last-row,~
10917   left-margin,~
10918   light-syntax,~
10919   light-syntax-expanded,~
10920   name,~

```

```

10921     no-cell-nodes,~
10922     nullify-dots,~
10923     pgf-node-code,~
10924     r,~
10925     renew-dots,~
10926     respect-arraystretch,~
10927     right-margin,~
10928     rounded-corners,~
10929     rules~(with~the~subkeys~'color'~and~'width'),~
10930     small,~
10931     t,~
10932     vl原因,~
10933     xdots/color,~
10934     xdots/shorten-start,~
10935     xdots/shorten-end,~
10936     xdots/shorten-and~
10937     xdots/line-style.
10938 }
10939 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10940 {
10941     Unknown~key.\\
10942     The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
10943     \{NiceTabular\}. \\
10944     That~key~will~be~ignored. \\
10945     \c_@@_available_keys_str
10946 }
10947 {
10948     The~available~keys~are~(in~alphabetic~order):~
10949     &-in-blocks,~
10950     ampersand-in-blocks,~
10951     b,~
10952     baseline,~
10953     c,~
10954     caption,~
10955     cell-space-bottom-limit,~
10956     cell-space-limits,~
10957     cell-space-top-limit,~
10958     code-after,~
10959     code-for-first-col,~
10960     code-for-first-row,~
10961     code-for-last-col,~
10962     code-for-last-row,~
10963     columns-width,~
10964     corners,~
10965     custom-line,~
10966     create-extra-nodes,~
10967     create-medium-nodes,~
10968     create-large-nodes,~
10969     extra-left-margin,~
10970     extra-right-margin,~
10971     first-col,~
10972     first-row,~
10973     hlines,~
10974     hvlines,~
10975     hvlines-except-borders,~
10976     label,~
10977     last-col,~
10978     last-row,~
10979     left-margin,~
10980     light-syntax,~
10981     light-syntax-expanded,~
10982     name,~
10983     no-cell-nodes,~

```

```

10984     notes~(several~subkeys),~
10985     nullify-dots,~
10986     pgf-node-code,~
10987     renew-dots,~
10988     respect-arraystretch,~
10989     right-margin,~
10990     rounded-corners,~
10991     rules~(with~the~subkeys~'color'~and~'width'),~
10992     short-caption,~
10993     t,~
10994     tabularnote,~
10995     vl原因,~
10996     xdots/color,~
10997     xdots/shorten-start,~
10998     xdots/shorten-end,~
10999     xdots/shorten-and~
11000     xdots/line-style.
11001 }
11002 \@@_msg_new:nnn { Duplicate~name }
11003 {
11004     Duplicate~name.\\
11005     The~name~' \l_keys_value_tl '~is~already~used~and~you~shouldn't~use~
11006     the~same~environment~name~twice.~You~can~go~on,~but,~
11007     maybe,~you~will~have~incorrect~results~especially~
11008     if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
11009     message~again,~use~the~key~'allow-duplicate-names'~in~
11010     ' \token_to_str:N \NiceMatrixOptions '~.\\
11011     \bool_if:NF \g_@@_messages_for_Overleaf_bool
11012     { For~a~list~of~the~names~already~used,~type~H~<return>. }
11013 }
11014 {
11015     The~names~already~defined~in~this~document~are:~
11016     \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
11017 }
11018 \@@_msg_new:nn { caption-above~in~env }
11019 {
11020     The~key~'caption-above'~must~be~used~in~\token_to_str:N \NiceMatrixOptions.\\
11021     That~key~will~be~ignored.
11022 }
11023 \@@_msg_new:nn { show-cell-names }
11024 {
11025     There~is~no~key~'show-cell-names'~in~nicematrix.\\
11026     You~should~use~the~command~\token_to_str:N \ShowCellNames\\
11027     in~the~\token_to_str:N \CodeBefore\\ or~the~\token_to_str:N
11028     \CodeAfter. \\
11029     That~key~will~be~ignored.
11030 }
11031 \@@_msg_new:nn { Option~auto~for~columns-width }
11032 {
11033     Erroneous~use.\\
11034     You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
11035     That~key~will~be~ignored.
11036 }
11037 \@@_msg_new:nn { NiceTabularX~without~X }
11038 {
11039     NiceTabularX~without~X.\\
11040     You~should~not~use~\{NiceTabularX\}~without~X~columns.\\
11041     However,~you~can~go~on.
11042 }
11043 \@@_msg_new:nn { Preamble~forgotten }
11044 {

```

```

11045     Preamble~forgotten.\\
11046     You~have~probably~forgotten~the~preamble~of~your~
11047     \@@_full_name_env: . \\
11048     This~error~is~fatal.
11049 }
11050 \@@_msg_new:nn { Invalid~col~number }
11051 {
11052     Invalid~column~number.\\
11053     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11054     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.
11055 }
11056 \@@_msg_new:nn { Invalid~row~number }
11057 {
11058     Invalid~row~number.\\
11059     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11060     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.
11061 }
11062 \@@_define_com:NNN p ( )
11063 \@@_define_com:NNN b [ ]
11064 \@@_define_com:NNN v | |
11065 \@@_define_com:NNN V \l \l
11066 \@@_define_com:NNN B \{ \}

```

Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	3
4	Parameters	9
5	The command <code>\tabularnote</code>	20
6	Command for creation of rectangle nodes	25
7	The options	26
8	Important code used by <code>{NiceArrayWithDelims}</code>	37
9	The <code>\CodeBefore</code>	51
10	The environment <code>{NiceArrayWithDelims}</code>	55
11	Construction of the preamble of the array	61
12	The redefinition of <code>\multicolumn</code>	77
13	The environment <code>{NiceMatrix}</code> and its variants	95
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	96
15	After the construction of the array	97
16	We draw the dotted lines	104
17	The actual instructions for drawing the dotted lines with Tikz	119
18	User commands available in the new environments	125
19	The command <code>\line</code> accessible in code-after	131
20	The command <code>\RowStyle</code>	133
21	Colors of cells, rows and columns	136
22	The vertical and horizontal rules	148
23	The empty corners	165
24	The environment <code>{NiceMatrixBlock}</code>	167
25	The extra nodes	169
26	The blocks	173
27	How to draw the dotted lines transparently	200
28	Automatic arrays	200
29	The redefinition of the command <code>\dotfill</code>	202
30	The command <code>\diagbox</code>	202

31	The keyword <code>\CodeAfter</code>	203
32	The delimiters in the preamble	204
33	The command <code>\SubMatrix</code>	205
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	214
35	The commands <code>HBrace</code> et <code>VBrace</code>	217
36	The command <code>TikzEveryCell</code>	220
37	The command <code>\ShowCellNames</code>	222
38	We process the options at package loading	223
39	About the package underscore	225
40	Error messages of the package	225