

The code of the package **nicematrix**^{*}

F. Pantigny
fpantigny@wanadoo.fr

December 14, 2025

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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 7.5 of **nicematrix**, at the date of 2025/12/14.

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 currying.

```

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 programmation. 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:n { 1 pt } { \hbox { . } }
132     \mkern 2 mu
133     \box_move_up:n { 4 pt } { \hbox { . } }
134     \mkern 2 mu
135     \box_move_up:n { 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:NTF \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 `1` 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 raise then 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 contain 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.

```
338 \bool_new:N \g_@@_aux_found_bool
```

In particuler, 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_t1` 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_t1
```

The token list `\g_@@_array_preamble_t1` will contain the preamble constructed by `nicematrix` for the environment `{array}` (of array).

```
343 \tl_new:N \g_@@_array_preamble_t1
```

For `\multicolumn`.

```
344 \tl_new:N \g_@@_preamble_t1
```

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_t1
```

```
346 \str_set:Nn \l_@@_columns_type_t1 { 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_t1
```

```
348 \tl_new:N \l_@@_xdots_up_t1
```

```
349 \tl_new:N \l_@@_xdots_middle_t1
```

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 \g_@@_test_if_math_mode:
352 {
353   \if_mode_math: \else:
354     \g_@@_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_t1
```

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_t1
```

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_t1
374 \tl_new:N \g_nicematrix_code_before_t1
```

The value of the key `code-before` will be added to the left of `\g_@@_pre_code_before_t1`. 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_t1
376 \tl_new:N \g_nicematrix_code_after_t1
```

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_t1
```

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 `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  }
```

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 \captype } { \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       \NewDocumentCommand \tabularnote { o m }
574           { \err_enumitem_not_loaded: }
575   }
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:N \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:nnnn #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         {
694             rectangle
695             {
696                 center
697                 {
698                     \vbox_to_ht:nn
699                     {
700                         \dim_abs:n { #5 - #3 }
701                         {
702                             \vfill
703                             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
704                         }
705                     }
706                 }
707             }
708         }
709     \end{pgfscope}
710 }

```

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

```

823   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
824   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
825   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 `\Idots`, will be caught when `\Ddots` or `\Idots` is used (during the construction of the array and not when we draw the dotted lines).

```

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

830 \keys_define:nn { nicematrix / rules }
831 {
832   color .tl_set:N = \l_@@_rules_color_tl ,
833   color .value_required:n = true ,
834   width .dim_set:N = \arrayrulewidth ,
835   width .value_required:n = true ,
836   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
837 }

```

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

```

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

```

```

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

```

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

```

902     { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
903     }
904     { \@@_error:n { One-letter~allowed } }
905     },
906     vlines-in-sub-matrix .value_required:n = true ,
907     hvlines .code:n =
908     {
909         \bool_set_true:N \l_@@_hvlines_bool
910         \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
911         \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
912     },
913     hvlines .value_forbidden:n = true ,
914     hvlines-except-borders .code:n =
915     {
916         \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
917         \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
918         \bool_set_true:N \l_@@_hvlines_bool
919         \bool_set_true:N \l_@@_except_borders_bool
920     },
921     hvlines-except-borders .value_forbidden:n = true ,
922     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.

```

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

```

```

942     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
943     respect_arraystretch .value_forbidden:n = true ,
944     pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
945     pgf-node-code .value_required:n = true
946 }

```

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

```

947 \keys_define:nn { nicematrix / environments }
948 {
949     corners .clist_set:N = \l_@@_corners_clist ,
950     corners .default:n = { NW , SW , NE , SE } ,
951     code-before .code:n =
952     {
953         \tl_if_empty:nF { #1 }
954         {
955             \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
956             \bool_set_true:N \l_@@_code_before_bool
957         }
958     },
959     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}`.

```

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

```

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

```

971 \legacy_if:nF { measuring@ }
972 {
973     \str_set:Ne \l_@@_name_str { #1 }
974     \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
975     { \@@_err_duplicate_names:n { #1 } }
976     { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
977 },
978 name .value_required:n = true ,
979 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
980 code-after .value_required:n = true ,
981 }

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

984 \keys_define:nn { nicematrix / notes }
985 {
986     para .bool_set:N = \l_@@_notes_para_bool ,
987     para .default:n = true ,
988     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
989     code-before .value_required:n = true ,
990     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
991     code-after .value_required:n = true ,

```

```

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

```

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

```

1030 \keys_define:nn { nicematrix }
1031 {
1032     NiceMatrixOptions .inherit:n =
1033         { nicematrix / Global } ,
1034     NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1035     NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1036     NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1037     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1038     SubMatrix / rules .inherit:n = nicematrix / rules ,
1039     CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1040     CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1041     CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1042     NiceMatrix .inherit:n =
1043     {
1044         nicematrix / Global ,
1045         nicematrix / environments ,
1046     },
1047     NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1048     NiceMatrix / rules .inherit:n = nicematrix / rules ,
1049     NiceTabular .inherit:n =
1050     {

```

```

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

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

```

1072 \keys_define:nn { nicematrix / NiceMatrixOptions }
1073 {
1074   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1075   delimiters / color .value_required:n = true ,
1076   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1077   delimiters / max-width .default:n = true ,
1078   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1079   delimiters .value_required:n = true ,
1080   width .dim_set:N = \l_@@_width_dim ,
1081   width .value_required:n = true ,
1082   last-col .code:n =
1083     \tl_if_empty:nF { #1 }
1084     { \@@_error:n { last-col~non~empty~for~NiceMatrixOptions } }
1085     \int_zero:N \l_@@_last_col_int ,
1086   small .bool_set:N = \l_@@_small_bool ,
1087   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.

```

1088   renew-matrix .code:n = \@@_renew_matrix: ,
1089   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}`.

```

1090   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.

```

1091   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)`.

```

1092   \str_if_eq:eeTF { #1 } { auto }
1093   { \@@_error:n { Option~auto~for~columns-width } }
1094   { \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.

```

1095     allow-duplicate-names .code:n =
1096         \cs_set:Nn \@@_err_duplicate_names:n { } ,
1097     allow-duplicate-names .value_forbidden:n = true ,
1098     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1099     notes .value_required:n = true ,
1100     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1101     sub-matrix .value_required:n = true ,
1102     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1103     matrix / columns-type .value_required:n = true ,
1104     caption-above .bool_set:N = \l_@@_caption_above_bool ,
1105     caption-above .default:n = true ,
1106     unknown .code:n =
1107         \@@_unknown_key:nn
1108             { nicematrix / Global , nicematrix / NiceMatrixOptions }
1109             { Unknown-key~for~NiceMatrixOptions }
1110
1111 }
```

`\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.

```

1111 \NewDocumentCommand \NiceMatrixOptions { m }
1112     { \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.

```

1113 \keys_define:nn { nicematrix / NiceMatrix }
1114 {
1115     last-col .code:n = \tl_if_empty:nTF { #1 }
1116         {
1117             \bool_set_true:N \l_@@_last_col_without_value_bool
1118             \int_set:Nn \l_@@_last_col_int { -1 }
1119         }
1120         { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1121     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1122     columns-type .value_required:n = true ,
1123     l .meta:n = { columns-type = l } ,
1124     r .meta:n = { columns-type = r } ,
1125     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1126     delimiters / color .value_required:n = true ,
1127     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1128     delimiters / max-width .default:n = true ,
1129     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1130     delimiters .value_required:n = true ,
1131     small .bool_set:N = \l_@@_small_bool ,
1132     small .value_forbidden:n = true ,
1133     unknown .code:n = \@@_error:n { Unknown-key~for~NiceMatrix }
1134 }
```

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

```

1135 \keys_define:nn { nicematrix / NiceArray }
1136 {
```

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.

```

1137     small .bool_set:N = \l_@@_small_bool ,
1138     small .value_forbidden:n = true ,
1139     last-col .code:n = \tl_if_empty:nF { #1 }
1140             { \@@_error:n { last-col~non~empty~for~NiceArray } }
```

```

1141           \int_zero:N \l_@@_last_col_int ,
1142   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1143   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1144   unknown .code:n =
1145     \@@_unknown_key:nn
1146     { nicematrix / NiceArray , nicematrix / Global , nicematrix / environments}
1147     { Unknown-key-for-NiceArray }
1148 }

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

```

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

```

1171 \keys_define:nn { nicematrix / NiceTabular }
1172 {

```

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.

```

1173   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1174     \bool_set_true:N \l_@@_width_used_bool ,
1175   width .value_required:n = true ,
1176   notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1177   tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1178   tabularnote .value_required:n = true ,
1179   caption .tl_set:N = \l_@@_caption_tl ,
1180   caption .value_required:n = true ,
1181   short-caption .tl_set:N = \l_@@_short_caption_tl ,
1182   short-caption .value_required:n = true ,
1183   label .tl_set:N = \l_@@_label_tl ,
1184   label .value_required:n = true ,
1185   last-col .code:n = \tl_if_empty:nF { #1 }
1186     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1187     \int_zero:N \l_@@_last_col_int ,
1188   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1189   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1190   unknown .code:n =
1191     \@@_unknown_key:nn
1192     { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
1193     { Unknown-key-for-NiceTabular }
1194 }

```

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

1195 \keys_define:nn { nicematrix / CodeAfter }
1196 {
1197   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1198   delimiters / color .value_required:n = true ,
1199   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1200   rules .value_required:n = true ,
1201   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1202   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1203   sub-matrix .value_required:n = true ,
1204   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1205 }
```

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}`).

```
1206 \cs_new_protected:Npn \@@_cell_begin:
1207 {
```

`\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.

```
1208 \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).

```
1209 \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.

```
1210 \cs_set_eq:NN \Hline \@@_Hline_in_cell:
```

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

```
1211 \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.

```
1212 \int_compare:nNnT { \c@jCol } = { \c_one_int }
1213 {
1214   \int_compare:nNnT { \l_@@_first_col_int } = { \c_one_int }
1215   { \@@_begin_of_row: }
1216 }
```

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::`

```
1217 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1218 \@@_tuning_not_tabular_begin:
1219 \@@_tuning_first_row:
1220 \@@_tuning_last_row:
1221 \g_@@_row_style_tl
1222 }
```

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.

```
1223 \cs_new_protected:Npn \@@_tuning_first_row:
1224 {
1225     \if_int_compare:w \c@iRow = \c_zero_int
1226         \if_int_compare:w \c@jCol > \c_zero_int
1227             \l_@@_code_for_first_row_tl
1228             \xglobal \colorlet{nicematrix-first-row}{.}
1229         \fi:
1230     \fi:
1231 }
```

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

```
\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.

```
1232 \cs_new_protected:Npn \@@_tuning_last_row:
1233 {
1234     \if_int_compare:w \c@iRow = \l_@@_last_row_int
1235         \l_@@_code_for_last_row_tl
1236         \xglobal \colorlet{nicematrix-last-row}{.}
1237     \fi:
1238 }
```

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

```
1239 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```
1240 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1241 {
1242     \m@th
1243     $ % $
```

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

```
1244 \@@_tuning_key_small:
1245 }
1246 \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.

```

1247 \cs_new_protected:Npn \@@_begin_of_row:
1248 {
1249     \int_gincr:N \c@iRow
1250     \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1251     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \arstrutbox }
1252     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \arstrutbox }
1253     \pgfpicture
1254     \pgfrememberpicturepositiononpagetrue
1255     \pgfcoordinate
1256         { \@@_env: - row - \int_use:N \c@iRow - base }
1257         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1258     \str_if_empty:NF \l_@@_name_str
1259     {
1260         \pgfnodealias
1261             { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1262             { \@@_env: - row - \int_use:N \c@iRow - base }
1263     }
1264     \endpgfpicture
1265 }
```

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.

```

1266 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1267 {
1268     \int_if_zero:nTF { \c@iRow }
1269     {
1270         \dim_compare:nNnT
1271             { \box_dp:N \l_@@_cell_box } > { \g_@@_dp_row_zero_dim }
1272             { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1273         \dim_compare:nNnT
1274             { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_zero_dim }
1275             { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1276     }
1277     {
1278         \int_compare:nNnT { \c@iRow } = { \c_one_int }
1279         {
1280             \dim_compare:nNnT
1281                 { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_one_dim }
1282                 { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1283         }
1284     }
1285 }
```

```

1286 \cs_new_protected:Npn \@@_rotate_cell_box:
1287 {
1288     \box_rotate:Nn \l_@@_cell_box { 90 }
1289     \bool_if:NTF \g_@@_rotate_c_bool
1290     {
1291         \hbox_set:Nn \l_@@_cell_box
1292         {
1293             \m@th
1294             $ % $
1295             \vcenter { \box_use:N \l_@@_cell_box }
1296             $ % $
1297         }
1298     }
1299 }
```

```

1300 \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
1301 {
1302     \vbox_set_top:Nn \l_@@_cell_box
1303     {
1304         \vbox_to_zero:n {}
1305         \skip_vertical:n { - \box_ht:N \carstrutbox + 0.8 ex }
1306         \box_use:N \l_@@_cell_box
1307     }
1308 }
1309 }
1310 \bool_gset_false:N \g_@@_rotate_bool
1311 \bool_gset_false:N \g_@@_rotate_c_bool
1312 }

1313 \cs_new_protected:Npn \@@_adjust_size_box:
1314 {
1315     \dim_compare:nNnT { \g_@@_blocks_wd_dim } > { \c_zero_dim }
1316     {
1317         \box_set_wd:Nn \l_@@_cell_box
1318         { \dim_max:nn { \box_wd:N \l_@@_cell_box } { \g_@@_blocks_wd_dim } }
1319         \dim_gzero:N \g_@@_blocks_wd_dim
1320     }
1321     \dim_compare:nNnT { \g_@@_blocks_dp_dim } > { \c_zero_dim }
1322     {
1323         \box_set_dp:Nn \l_@@_cell_box
1324         { \dim_max:nn { \box_dp:N \l_@@_cell_box } { \g_@@_blocks_dp_dim } }
1325         \dim_gzero:N \g_@@_blocks_dp_dim
1326     }
1327     \dim_compare:nNnT { \g_@@_blocks_ht_dim } > { \c_zero_dim }
1328     {
1329         \box_set_ht:Nn \l_@@_cell_box
1330         { \dim_max:nn { \box_ht:N \l_@@_cell_box } { \g_@@_blocks_ht_dim } }
1331         \dim_gzero:N \g_@@_blocks_ht_dim
1332     }
1333 }

1334 \cs_new_protected:Npn \@@_cell_end:
1335 {

```

The following command is nullified in the tabulars.

```

1336 \@@_tuning_not_tabular_end:
1337 \hbox_set_end:
1338 \@@_cell_end_i:
1339 }

1340 \cs_new_protected:Npn \@@_cell_end_i:
1341 {

```

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.

```

1342 \g_@@_cell_after_hook_tl
1343 \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
1344 \@@_adjust_size_box:

1345 \box_set_ht:Nn \l_@@_cell_box
1346     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1347 \box_set_dp:Nn \l_@@_cell_box
1348     { \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”).

```

1349 \@@_update_max_cell_width:

```

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

```

1350 \@@_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.

```

1351 \bool_if:NTF \g_@@_empty_cell_bool
1352   { \box_use_drop:N \l_@@_cell_box }
1353   {
1354     \bool_if:NTF \g_@@_not_empty_cell_bool
1355       { \@@_print_node_cell: }
1356       {
1357         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
1358           { \@@_print_node_cell: }
1359           { \box_use_drop:N \l_@@_cell_box }
1360       }
1361     }
1362   \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
1363     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1364   \bool_gset_false:N \g_@@_empty_cell_bool
1365   \bool_gset_false:N \g_@@_not_empty_cell_bool
1366 }
```

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

```

1367 \cs_new_protected:Npn \@@_update_max_cell_width:
1368   {
1369     \dim_gset:Nn \g_@@_max_cell_width_dim
1370       { \dim_max:nn { \g_@@_max_cell_width_dim } { \box_wd:N \l_@@_cell_box } }
1371 }
```

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`).

```

1372 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1373   {
1374     \@@_math_toggle:
1375     \hbox_set_end:
1376     \bool_if:NF \g_@@_rotate_bool
1377     {
1378       \hbox_set:Nn \l_@@_cell_box
1379       {
1380         \makebox [ \l_@@_col_width_dim ] [ s ]
1381           { \hbox_unpack_drop:N \l_@@_cell_box }
1382       }
1383     }
1384     \@@_cell_end_i:
1385 }
```

```

1386 \pgfset
1387 {
1388     nicematrix / cell-node /.style =
1389     {
1390         inner sep = \c_zero_dim ,
1391         minimum width = \c_zero_dim
1392     }
1393 }

```

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.

```

1394 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1395 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1396 {
1397     \use:c
1398     {
1399         __siunitx_table_align_
1400         \bool_if:NTF \l_siunitx_table_text_bool
1401             { \l_siunitx_table_align_text_tl }
1402             { \l_siunitx_table_align_number_tl }
1403     :n
1404 }
1405 { #1 }
1406 }

```

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!

```

1407 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1408 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1409 {
1410     \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1411     \hbox:n
1412     {
1413         \pgfsys@markposition
1414             { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1415     }
1416 #1
1417 \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1418     \hbox:n
1419     {
1420         \pgfsys@markposition
1421             { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1422     }
1423 }

1424 \cs_new_protected:Npn \@@_print_node_cell:
1425 {
1426     \socket_use:nn { nicematrix / siunitx-wrap }
1427     { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1428 }

```

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

```

1429 \cs_new_protected:Npn \@@_node_cell:
1430 {
1431     \pgfpicture
1432     \pgfsetbaseline \c_zero_dim

```

```

1433 \pgfrememberpicturepositiononpagetrue
1434 \pgfset { nicematrix / cell-node }
1435 \pgfnode
1436 { rectangle }
1437 { base }
1438 {

```

The following instruction `\set@color` has been added on 2022/10/06. It's necessary only with XeLaTeX and not with the other engines (we don't know why).

```

1439 \sys_if_engine_xetex:T { \set@color }
1440 \box_use:N \l_@@_cell_box
1441 }
1442 { \l_@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1443 { \l_@@_pgf_node_code_tl }
1444 \str_if_empty:NF \l_@@_name_str
1445 {
1446 \pgfnodealias
1447 { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1448 { \l_@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1449 }
1450 \endpgfpicture
1451 }

```

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}

```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & \cdots & & 6 \\ 7 & \cdots & & \text{red dots} \end{pmatrix}$$

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).

```

1452 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1453 {
1454 \bool_if:nTF { #1 } { \tl_gput_left:ce } { \tl_gput_right:ce }
1455 { \g_@@_#2 _ lines _ tl }
1456 {
1457 \use:c { @@ _ draw _ #2 : nnn }
1458 { \int_use:N \c@iRow }
1459 { \int_use:N \c@jCol }
1460 { \exp_not:n { #3 } }
1461 }
1462 }

1463 \cs_new_protected:Npn \@@_array:n
1464 {
1465 \dim_set:Nn \col@sep
1466 { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1467 \dim_compare:nNnTF { \l_@@_tabular_width_dim } = { \c_zero_dim }
1468 { \def \halignto { } }
1469 { \cs_set_nopar:Npe \halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1470     \@tabarray
\l_@@_baseline_tl may have the value t, c or b. However, if the value is b, we compose
the \array (of array) with the option t and the right translation will be done further. Re-
mark that \str_if_eq:eeTF is fully expandable and we need something fully expandable here.
\str_if_eq:ee(TF) is faster than \str_if_eq:nn(TF).
1471     [ \str_if_eq:eeTF { \l_@@_baseline_tl } { c } { c } { t } ]
1472   }
1473 \cs_generate_variant:Nn \@@_array { 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`.

```

1474 \bool_if:NTF \c_@@_revtex_bool
1475   { \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`.

```

1476   { \cs_set_eq:cN { @@_old_ar@ialign: } \ar@ialign }
```

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

```

1477 \cs_new_protected:Npn \@@_create_row_node:
1478   {
1479     \int_compare:nNnT { \c@iRow } > { \g_@@_last_row_node_int }
1480     {
1481       \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1482       \@@_create_row_node_i:
1483     }
1484   }
1485 \cs_new_protected:Npn \@@_create_row_node_i:
1486   {
```

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

```

1487 \hbox
1488   {
1489     \bool_if:NT \l_@@_code_before_bool
1490     {
1491       \vtop
1492       {
1493         \skip_vertical:N 0.5\arrayrulewidth
1494         \pgfsys@markposition
1495         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1496         \skip_vertical:N -0.5\arrayrulewidth
1497       }
1498     }
1499     \pgfpicture
1500     \pgfrememberpictureonpagetrue
1501     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1502     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1503     \str_if_empty:NF \l_@@_name_str
1504     {
1505       \pgfnodealias
1506       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1507       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1508     }
1509     \endpgfpicture
1510   }
1511 }
```



```

1512 \cs_new_protected:Npn \@@_in_everycr:
1513   {
1514     \tbl_if_row_was_started:T { \UseTaggingSocket { \tbl / row / end } }
```

```

1515 \tbl_update_cell_data_for_next_row:
1516 \int_gzero:N \c@jCol
1517 \bool_gset_false:N \g_@@_after_col_zero_bool
1518 \bool_if:NF \g_@@_row_of_col_done_bool
1519 {
1520     \@@_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).

```

1521 \clist_if_empty:NF \l_@@_hlines_clist
1522 {
1523     \str_if_eq:eeF { \l_@@_hlines_clist } { all }
1524     {
1525         \clist_if_in:NeT
1526             \l_@@_hlines_clist
1527             { \int_eval:n { \c@iRow + 1 } }
1528     }
1529     {

```

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.

```

1530     \int_compare:nNnT { \c@iRow } > { -1 }
1531     {
1532         \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
1533             { \hrule height \arrayrulewidth width \c_zero_dim }
1534     }
1535 }
1536 }
1537 }
1538 }
```

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

```

1539 \cs_set_protected:Npn \@@_renew_dots:
1540 {
1541     \cs_set_eq:NN \ldots \@@_Ldots:
1542     \cs_set_eq:NN \cdots \@@_Cdots:
1543     \cs_set_eq:NN \vdots \@@_Vdots:
1544     \cs_set_eq:NN \ddots \@@_Ddots:
1545     \cs_set_eq:NN \iddots \@@_Iddots:
1546     \cs_set_eq:NN \dots \@@_Ldots:
1547     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1548 }
```

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 redefinition⁵.

```

1549 \hook_gput_code:nnn { begindocument } { . }
1550 {
1551     \IfPackageLoadedTF { booktabs }
1552     {
1553         \cs_new_protected:Npn \@@_patch_booktabs:
1554             { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1555     }
1556     { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1557 }
```

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`.

```

1558 \cs_new_protected:Npn \@@_some_initialization:
1559 {
1560     \@@_everycr:
1561     \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1562     \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1563     \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1564     \dim_gzero:N \g_@@_dp_ante_last_row_dim
1565     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1566     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1567 }
```

`\@@_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`.

```

1568 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1569 {
```

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.

```

1570     \% \seq_gset_eq:NN \g_@@_pos_of_blocks_seq \g_@@_future_pos_of_blocks_seq
1571     \% \seq_gclear:N \g_@@_future_pos_of_blocks_seq
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
```

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

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     {
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             { \c@env: - row - \int_use:N \c@iRow - base }
1595         }
1596         \socket_assign_plugin: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 programmation, 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 `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 `\tabularnote` in the caption specified by the key `caption` and if that caption has to be composed above the tabular, we have now that information because it has been written in the `aux` file at a previous run. We use that information to start counting the tabular notes in the main array at the right value (we remember that the caption will be composed *after* the array!).

```

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@tabularnote { \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 \o@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 \c@iRow }
1701     \int_gzero_new:N \c@iRow
1702     \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol \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, you recreate the diagonal nodes by using the `row` nodes and the `col` nodes.

```
1778 \@@_create_diag_nodes:
```

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

```

1779 \bool_if:NT \g_@@_create_cell_nodes_bool { \@@_recreate_cell_nodes: }
1780 \endpgfpicture

```

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     \cs_set_eq:NN \cellcolor \@@_cellcolor
1790     \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1791     \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1792     \cs_set_eq:NN \rowcolor \@@_rowcolor
1793     \cs_set_eq:NN \rowcolors \@@_rowcolors
1794     \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1795     \cs_set_eq:NN \arraycolor \@@_arraycolor
1796     \cs_set_eq:NN \columncolor \@@_columncolor
1797     \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1798     \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1799     \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1800     \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1801     \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1802     \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1803 }
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:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1825     { \@@_rescan_for_spanish:N \l_@@_code_before_tl }
1826     \exp_last_unbraced:No \@@_CodeBefore_keys:
1827         \g_@@_pre_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`.

```

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: { O{ } }
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:NT \g_@@_aux_found_bool
1850     {
1851         \@@_pre_code_before:
1852         \legacy_if:nF { measuring@ } { #1 }
1853     }
1854 }

```

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.

```

1855 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1856 {
1857     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
1858     {
1859         \pgfposition { \@@_env: - ##1 - base } \@@_node_position:
1860         \pgfcoordinate { \@@_env: - row - ##1 - base }
1861             { \pgfpointdiff \@@_picture_position: \@@_node_position: }

```

```

1862 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
1863 {
1864     \cs_if_exist:cT
1865         { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - ####1 - NW }
1866         {
1867             \pgfsys@getposition
1868                 { \@@_env: - ##1 - ####1 - NW }
1869                 \@@_node_position:
1870             \pgfsys@getposition
1871                 { \@@_env: - ##1 - ####1 - SE }
1872                 \@@_node_position_i:
1873             \@@_pgf_rect_node:nnn
1874                 { \@@_env: - ##1 - ####1 }
1875                 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1876                 { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1877             }
1878         }
1879     }
1880 \@@_create_extra_nodes:
1881 \@@_create_aliases_last:
1882 }

1883 \cs_new_protected:Npn \@@_create_aliases_last:
1884 {
1885     \int_step_inline:nn { \c@iRow }
1886     {
1887         \pgfnodealias
1888             { \@@_env: - ##1 - last }
1889             { \@@_env: - ##1 - \int_use:N \c@jCol }
1890     }
1891     \int_step_inline:nn { \c@jCol }
1892     {
1893         \pgfnodealias
1894             { \@@_env: - last - ##1 }
1895             { \@@_env: - \int_use:N \c@iRow - ##1 }
1896     }
1897     \pgfnodealias
1898         { \@@_env: - last - last }
1899         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1900 }

```



```

1901 \cs_new_protected:Npn \@@_create_blocks_nodes:
1902 {
1903     \pgfpicture
1904     \pgf@relevantforpicturesizefalse
1905     \pgfrememberpicturepositiononpagetrue
1906     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1907         { \@@_create_one_block_node:nnnnn ##1 }
1908     \endpgfpicture
1909 }

```

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

```

1910 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1911 {
1912     \tl_if_empty:nF { #5 }
1913     {
1914         \@@_qpoint:n { col - #2 }
1915         \dim_set_eq:NN \l_tmpa_dim \pgf@x

```

⁷Moreover, there is also in the list `\g_@@_pos_of_blocks_seq` the positions of the dotted lines (created by `\Cdots`, etc.) and, for these entries, there is, of course, no name (the fifth component is empty).

```

1916     \@@_qpoint:n { #1 }
1917     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1918     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1919     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1920     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1921     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1922     \@@_pgf_rect_node:nnnn
1923         { \@@_env: - #5 }
1924         { \dim_use:N \l_tmpa_dim }
1925         { \dim_use:N \l_tmpb_dim }
1926         { \dim_use:N \l_@@_tmpc_dim }
1927         { \dim_use:N \l_@@_tmpd_dim }
1928     }
1929 }

1930 \cs_new_protected:Npn \@@_patch_for_revtex:
1931 {
1932     \cs_set_eq:NN \caddamp \caddamp@LaTeX
1933     \cs_set_eq:NN \carray \carray@array
1934     \cs_set_eq:NN \ctabular \ctabular@array
1935     \cs_set:Npn \tabarray { \ifnextchar [ { \carray } { \carray [ c ] } }
1936     \cs_set_eq:NN \array \array@array
1937     \cs_set_eq:NN \endarray \endarray@array
1938     \cs_set:Npn \endtabular { \endarray $ \egroup } % $
1939     \cs_set_eq:NN \mkpream \mkpream@array
1940     \cs_set_eq:NN \classx \classx@array
1941     \cs_set_eq:NN \insert@column \insert@column@array
1942     \cs_set_eq:NN \arraycr \arraycr@array
1943     \cs_set_eq:NN \xarraycr \xarraycr@array
1944     \cs_set_eq:NN \xargarraycr \xargarraycr@array
1945 }

```

10 The environment {NiceArrayWithDelims}

```

1946 \NewDocumentEnvironment { NiceArrayWithDelims }
1947     { m m O { } m ! O { } t \CodeBefore }
1948     {
1949         \bool_if:NT \c_@@_revtex_bool { \@@_patch_for_revtex: }
1950         \@@_provide_pgfsyspdfmark:
1951         \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.

```

1952 \bgroup
1953     \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1954     \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1955     \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1956     \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

1957     \int_gzero:N \g_@@_block_box_int
1958     \dim_gzero:N \g_@@_width_last_col_dim
1959     \dim_gzero:N \g_@@_width_first_col_dim
1960     \bool_gset_false:N \g_@@_row_of_col_done_bool
1961     \str_if_empty:NT \g_@@_name_env_str
1962         { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1963     \bool_if:NTF \l_@@_tabular_bool
1964         { \mode_leave_vertical: }
1965         { \@@_test_if_math_mode: }
1966     \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1967     \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.

```
1968     \cs_gset_eq:cN { @@_old_Carc@ } \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 `\tikzexternalisable` and not with `\tikzset{external/export=false}` which is *not* equivalent.

```
1969     \cs_if_exist:NT \tikz@library@external@loaded
1970     {
1971         \tikzexternalisable
1972         \cs_if_exist:NT \ifstandalone
1973             { \tikzset { external / optimize = false } }
1974     }
```

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

```
1975     \int_gincr:N \g_@@_env_int
1976     \bool_if:NF \l_@@_block_auto_columns_width_bool
1977     { \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.

```
1978     \seq_gclear:N \g_@@_blocks_seq
1979     \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`.

```
1980     \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1981     \seq_gclear:N \g_@@_pos_of_xdots_seq
1982     \tl_gclear_new:N \g_@@_code_before_tl
1983     \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.

```
1984     \tl_if_exist:cTF { g_@@_int_use:N \g_@@_env_int _ tl }
1985     {
1986         \bool_gset_true:N \g_@@_aux_found_bool
1987         \use:c { g_@@_int_use:N \g_@@_env_int _ tl }
1988     }
1989     { \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.

```
1990     \tl_gclear:N \g_@@_aux_tl
1991     \tl_if_empty:NF \g_@@_code_before_tl
1992     {
1993         \bool_set_true:N \l_@@_code_before_bool
1994         \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1995     }
1996     \tl_if_empty:NF \g_@@_pre_code_before_tl
1997     { \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`.

```
1998     \bool_if:NTF \g_@@_delims_bool
1999     { \keys_set:nn { nicematrix / pNiceArray } }
2000     { \keys_set:nn { nicematrix / NiceArray } }
2001     { #3 , #5 }
```

⁸e.g. `\color[rgb]{0.5,0.5,0}`

```
2002 \@@_set_CArc:o \l_@@_rules_color_t1 % 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`.

```
2003 \bool_if:nTF { #6 } { \@@_CodeBefore_Body:w } { \@@_pre_array: }
```

```
2004 }
```

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

```
2005 {
2006   \bool_if:NTF \l_@@_light_syntax_bool
2007   {
2008     \use:c { end @@-light-syntax }
2009     \use:c { end @@-normal-syntax }
2010   $ % $
2011   \skip_horizontal:N \l_@@_right_margin_dim
2012   \skip_horizontal:N \l_@@_extra_right_margin_dim
2013   \hbox_set_end:
2014   \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.

```
2014 \bool_if:NT \l_@@_width_used_bool
2015 {
2016   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2017   {
2018     \@@_error_or_warning:n { width-without-X-columns }
```

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

```
2019 \fp_compare:nNnT { \g_@@_total_X_weight_fp } > { \c_zero_fp }
2020   {
2021     \@@_compute_width_X: }
```

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

```
2021 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
2022 {
2023   \bool_if:NF \l_@@_last_row_without_value_bool
2024   {
2025     \int_compare:nNnF { \l_@@_last_row_int } = { \c@iRow }
2026     {
2027       \@@_error:n { Wrong-last-row }
2028       \int_set_eq:NN \l_@@_last_row_int \c@iRow
2029     }
2030   }
2031 }
```

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

```
2032 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2033 \bool_if:NTF \g_@@_last_col_found_bool
2034   {
2035     \int_gdecr:N \c@jCol
2036   }
2037   \int_compare:nNnT { \l_@@_last_col_int } > { -1 }
2038   {
2039     \@@_error:n { last-col-not-used }
```

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

```
2038 }
```

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

```
2039 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2040 \int_compare:nNnT { \l_@@_last_row_int } > { -1 }
2041   { \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).

```
2042 \int_if_zero:nT { \l_@@_first_col_int }
2043   { \skip_horizontal:N \g_@@_width_first_col_dim }
```

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

```
2044 \bool_if:nTF { ! \g_@@_delims_bool }
2045 {
2046   \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
2047     { \@@_use_arraybox_with_notes_c: }
2048   {
2049     \str_if_eq:eeTF { \l_@@_baseline_tl } { b }
2050       { \@@_use_arraybox_with_notes_b: }
2051       { \@@_use_arraybox_with_notes: }
2052   }
2053 }
```

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).

```
2054 {
2055   \int_if_zero:nTF { \l_@@_first_row_int }
2056   {
2057     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2058     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2059   }
2060   { \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

```
2061 \int_compare:nNnTF { \l_@@_last_row_int } > { -2 }
2062 {
2063   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2064   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2065 }
2066 { \dim_zero:N \l_tmpb_dim }
2067 \hbox_set:Nn \l_tmpa_box
2068 {
2069   \m@th
2070   $ % $
2071   \@@_color:o \l_@@_delimiters_color_tl
2072   \exp_after:wN \left \g_@@_left_delim_tl
2073   \vcenter
2074 }
```

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.

```
2075   \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2076   \hbox
2077   {
2078     \bool_if:NTF \l_@@_tabular_bool
2079       { \skip_horizontal:n { - \tabcolsep } }
2080       { \skip_horizontal:n { - \arraycolsep } }
2081     \@@_use_arraybox_with_notes_c:
2082     \bool_if:NTF \l_@@_tabular_bool
```

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

```

2083     { \skip_horizontal:n { - \tabcolsep } }
2084     { \skip_horizontal:n { - \arraycolsep } }
2085   }

```

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

```

2086     \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2087   }
2088   \exp_after:wN \right \g_@@_right_delim_tl
2089   $ % $
2090 }

```

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.

```

2091   \bool_if:NTF \l_@@_delimiters_max_width_bool
2092   {
2093     \@@_put_box_in_flow_bis:nn
2094     { \g_@@_left_delim_tl }
2095     { \g_@@_right_delim_tl }
2096   }
2097   \@@_put_box_in_flow:
2098 }

```

We take into account a potential “last column” (this “last column” has been constructed in an overlapping position and we have computed its width in `\g_@@_width_last_col_dim`: see p. 93).

```

2099   \bool_if:NT \g_@@_last_col_found_bool
2100   { \skip_horizontal:N \g_@@_width_last_col_dim }
2101   \bool_if:NT \l_@@_preamble_bool
2102   {
2103     \int_compare:nNnT { \c@jCol } < { \g_@@_static_num_of_col_int }
2104     { \@@_err_columns_not_used: }
2105   }
2106   \@@_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.

```

2107   \egroup

```

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

```

2108   \iow_now:Nn \mainaux { \ExplSyntaxOn }
2109   \iow_now:Nn \mainaux { \char_set_catcode_space:n { 32 } }
2110   \iow_now:Ne \mainaux
2111   {
2112     \tl_gclear_new:c { g_@@_int_use:N \g_@@_env_int _ tl }
2113     \tl_gset:cn { g_@@_int_use:N \g_@@_env_int _ tl }
2114     { \exp_not:o \g_@@_aux_tl }
2115   }
2116   \iow_now:Nn \mainaux { \ExplSyntaxOff }

2117   \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2118 }

```

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

```

2119   \cs_new_protected:Npn \@@_err_columns_not_used:
2120   {
2121     \@@_warning:n { columns-not-used }
2122     \cs_gset:Npn \@@_err_columns_not_used: { }
2123   }

```

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 .

```

2124 \cs_new_protected:Npn \@@_compute_width_X:
2125 {
2126     \tl_gput_right:Ne \g_@@_aux_tl
2127     {
2128         \bool_set_true:N \l_@@_X_columns_aux_bool
2129         \dim_set:Nn \l_@@_X_columns_dim
2130         {

```

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`

```

2131     \bool_lazy_and:nnTF
2132     { \g_@@_V_of_X_bool }
2133     { \l_@@_X_columns_aux_bool }
2134     { \dim_use:N \l_@@_X_columns_dim }
2135     {
2136         \dim_compare:nNnTF
2137         {
2138             \dim_abs:n
2139             { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2140         }
2141         <
2142         { 0.001 pt }
2143         { \dim_use:N \l_@@_X_columns_dim }
2144         {
2145             \dim_eval:n
2146             {
2147                 \l_@@_X_columns_dim
2148                 +
2149                 \fp_to_dim:n
2150                 {
2151                     (
2152                         \dim_eval:n
2153                         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2154                     )
2155                     / \fp_use:N \g_@@_total_X_weight_fp
2156                 }
2157             }
2158         }
2159     }
2160 }
2161 }
2162 }
```

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`.

```

2163 \cs_new_protected:Npn \@@_transform_preamble:
2164 {
2165     \@@_transform_preamble_i:
2166     \@@_transform_preamble_ii:
2167 }
2168 \cs_new_protected:Npn \@@_transform_preamble_i:
2169 {
2170     \int_gzero:N \c@jCol
```

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

```
2171     \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.
```

```
2172     \bool_gset_false:N \g_tmpb_bool
```

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

```
2173     \tl_gclear_new:N \g_@@_pre_cell_tl
```

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

```
2174     \int_zero:N \l_tmpa_int
2175     \tl_gclear:N \g_@@_array_preamble_tl
2176     \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2177     {
2178         \tl_gset:Nn \g_@@_array_preamble_tl
2179         { ! { \skip_horizontal:N \arrayrulewidth } }
2180     }
2181     {
2182         \clist_if_in:NnT \l_@@_vlines_clist 1
2183         {
2184             \tl_gset:Nn \g_@@_array_preamble_tl
2185             { ! { \skip_horizontal:N \arrayrulewidth } }
2186         }
2187     }
```

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

```
2188     \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2189     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
2190
2191     \@@_replace_columncolor:
2192 }
```

```
2192 \cs_new_protected:Npn \@@_transform_preamble_i:
2193 {
```

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

```
2194     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2195     {
2196         \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2197         { \bool_gset_true:N \g_@@_delims_bool }
2198     }
2199     { \bool_gset_true:N \g_@@_delims_bool }
```

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

```
2200     \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).

```
2201     \int_if_zero:nTF { \l_@@_first_col_int }
2202     { \tl_gput_left:N \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2203     {
2204         \bool_if:NF \g_@@_delims_bool
2205         {
2206             \bool_if:NF \l_@@_tabular_bool
2207             {
2208                 \clist_if_empty:NT \l_@@_vlines_clist
2209                 {
2210                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2211                     { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
```

```

2212         }
2213     }
2214   }
2215 }
2216 \int_compare:nNnTF { \l_@@_last_col_int } > { -1 }
2217   { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2218 {
2219   \bool_if:NF \g_@@_delims_bool
2220   {
2221     \bool_if:NF \l_@@_tabular_bool
2222     {
2223       \clist_if_empty:NT \l_@@_vlines_clist
2224       {
2225         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2226           { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2227       }
2228     }
2229   }
2230 }
```

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.

```

2231 \tag_if_active:F
2232 {
```

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

```

2233 \dim_compare:nNnT { \l_@@_tabular_width_dim } = { \c_zero_dim }
2234   {
2235     \tl_gput_right:Nn \g_@@_array_preamble_tl
2236       { > { \@@_err_too_many_cols: } 1 }
2237   }
2238 }
2239 }
```

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.

```

2240 \cs_new_protected:Npn \@@_rec_preamble:n #1
2241 {
```

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

```

2242   \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2243     { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2244 }
```

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

```

2245 \cs_if_exist:cTF { NC @ find @ #1 }
2246 {
2247   \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2248   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2249 }
2250 {
```

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`.

```

2251         \str_if_eq:nnTF { #1 } { S }
2252             { \@@_fatal:n { unknown~column~type-S } }
2253             { \@@_fatal:nn { unknown~column~type } { #1 } }
2254     }
2255 }
2256 }
```

For c, l and r

```

2257 \cs_new_protected:Npn \@@_c: #1
2258 {
2259     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2260     \tl_gclear:N \g_@@_pre_cell_tl
2261     \tl_gput_right:Nn \g_@@_array_preamble_tl
2262     { > \@@_cell_begin: c < \@@_cell_end: }
```

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

```

2263 \int_gincr:N \c@jCol
2264 \@@_rec_preamble_after_col:n
2265 }

2266 \cs_new_protected:Npn \@@_l: #1
2267 {
2268     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2269     \tl_gclear:N \g_@@_pre_cell_tl
2270     \tl_gput_right:Nn \g_@@_array_preamble_tl
2271     {
2272         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2273         l
2274         < \@@_cell_end:
2275     }
2276     \int_gincr:N \c@jCol
2277     \@@_rec_preamble_after_col:n
2278 }

2279 \cs_new_protected:Npn \@@_r: #1
2280 {
2281     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2282     \tl_gclear:N \g_@@_pre_cell_tl
2283     \tl_gput_right:Nn \g_@@_array_preamble_tl
2284     {
2285         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2286         r
2287         < \@@_cell_end:
2288     }
2289     \int_gincr:N \c@jCol
2290     \@@_rec_preamble_after_col:n
2291 }
```

For ! and @

```

2292 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2293 {
2294     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2295     \@@_rec_preamble:n
2296 }
2297 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }
```

For |

```

2298 \cs_new_protected:cpn { @@ _ | : } #1
2299 {
```

\l_tmpa_int is the number of successive occurrences of |

```

2300     \int_incr:N \l_tmpa_int
2301     \@@_make_preamble_i_i:n
2302 }
```

```

2303 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2304 {

```

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

```

2305   \str_if_eq:nnTF { #1 } { | }
2306   { \use:c { \@@_ | : } | }
2307   { \@@_make_preamble_i_ii:nn { } #1 }
2308 }

```

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

```

2309 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2310 {
2311   \str_if_eq:nnTF { #2 } { [ }
2312   { \@@_make_preamble_i_ii:nw { #1 } [ ]
2313   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2314 }
2315 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2316 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2317 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2318 {
2319   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2320   \tl_gput_right:Ne \g_@@_array_preamble_tl
2321   {

```

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

```

2322   \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2323   }
2324 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2325 {
2326   \@@_vline:n
2327   {
2328     position = \int_eval:n { \c@jCol + 1 } ,
2329     multiplicity = \int_use:N \l_tmpa_int ,
2330     total-width = \dim_use:N \l_@@_rule_width_dim ,
2331     #2
2332   }
2333 }

```

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.

```

2333   }
2334   \int_zero:N \l_tmpa_int
2335   \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2336   \@@_rec_preamble:n #1
2337 }

2338 \cs_new_protected:cpn { @<_> : } #1 #2
2339 {
2340   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2341   \@@_rec_preamble:n
2342 }
2343 \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.

```

2344 \keys_define:nn { nicematrix / p-column }
2345 {
2346   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2347   r .value_forbidden:n = true ,
2348   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2349   c .value_forbidden:n = true ,
2350   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,

```

```

2351     l .value_forbidden:n = true ,
2352     S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2353     S .value_forbidden:n = true ,
2354     p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2355     p .value_forbidden:n = true ,
2356     t .meta:n = p ,
2357     m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2358     m .value_forbidden:n = true ,
2359     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2360     b .value_forbidden:n = true
2361 }
```

For p but also b and m.

```

2362 \cs_new_protected:Npn \@@_p: #1
2363 {
2364     \str_set:Nn \l_@@_vpos_col_str { #1 }
```

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

```

2365     \@@_make_preamble_ii_i:n
2366 }
2367 \cs_set_eq:NN \@@_b: \@@_p:
2368 \cs_set_eq:NN \@@_m: \@@_p:
2369 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2370 {
2371     \str_if_eq:nnTF { #1 } { [ }
2372     { \@@_make_preamble_ii_ii:w [ }
2373     { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2374 }
2375 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2376 { \@@_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.

```

2377 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2378 {
```

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).

```

2379     \str_set:Nn \l_@@_hpos_col_str { j }
2380     \@@_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`.

```

2381     \setlength { \l_tmpa_dim } { #2 }
2382     \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2383 }
2384 \cs_new_protected:Npn \@@_keys_p_column:n #1
2385 { \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.

```

2386 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2387 {
```

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

```

2388     \use:e
2389     {
2390         \@@_make_preamble_ii_vi:nnnnnnn
2391         { \str_if_eq:eeTF { \l_@@_vpos_col_str } { p } { t } { b } }
2392         { #1 }
2393         {
```

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_t1` which will provide the horizontal alignment of the column to which belongs the cell.

```
2394     \str_if_eq:eeTF { \l_@@_hpos_col_str } { j }
2395         { \tl_clear:N \exp_not:N \l_@@_hpos_cell_t1 }
2396         { }
```

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

```
2397         \def \exp_not:N \l_@@_hpos_cell_t1
2398             { \str_lowercase:f { \l_@@_hpos_col_str } }
2399             }
2400         \IfPackageLoadedTF { ragged2e }
2401             {
2402                 \str_case:on \l_@@_hpos_col_str
2403                 { }
```

The following `\exp_not:N` are mandatory.

```
2404             c { \exp_not:N \Centering }
2405             l { \exp_not:N \RaggedRight }
2406             r { \exp_not:N \RaggedLeft }
2407             }
2408             }
2409             {
2410                 \str_case:on \l_@@_hpos_col_str
2411                     {
2412                         c { \exp_not:N \centering }
2413                         l { \exp_not:N \raggedright }
2414                         r { \exp_not:N \raggedleft }
2415                     }
2416                     }
2417                     #
2418                     }
2419                     { \str_if_eq:eeT { \l_@@_vpos_col_str } { m } \@@_center_cell_box: }
2420                     { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_begin:w }
2421                     { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_end: }
2422                     { #2 }
2423                     {
2424                         \str_case:onF \l_@@_hpos_col_str
2425                             {
2426                                 { j } { c }
2427                                 { si } { c }
2428                             }
```

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

```
2429             { \str_lowercase:f \l_@@_hpos_col_str }
2430             }
2431             }
```

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

```
2432     \int_gincr:N \c@jCol
2433     \@@_rec_preamble_after_col:n
2434 }
```

#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_t1` 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.
2435 \cs_new_protected:Npn \@@_make_preamble_i:nnnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2436 {
2437   \str_if_eq:eeTF { \l_@@_hpos_col_str } { si }
2438   {
2439     \tl_gput_right:Nn \g_@@_array_preamble_tl
2440     { > \@@_test_if_empty_for_S: }
2441   }
2442   {
2443     \str_if_eq:eeTF { #7 } { varwidth }
2444     {
2445       \tl_gput_right:Nn \g_@@_array_preamble_tl
2446       { > \@@_test_if_empty_varwidth: }
2447     }
2448     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2449   }
2450 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2451 \tl_gclear:N \g_@@_pre_cell_tl
2452 \tl_gput_right:Nn \g_@@_array_preamble_tl
2453   {
2454     > {

```

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.

```

2455   \dim_set:Nn \l_@@_col_width_dim { #2 }
2456   \@@_cell_begin:

```

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

```

2457   \use:c { #7 } [ #1 ] { #2 }

```

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

```

2458   \everypar
2459   {
2460     \vrule height \box_ht:N \carstrutbox width \c_zero_dim
2461     \everypar { }
2462   }

```

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

```

2463   #3

```

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

```

2464   \g_@@_row_style_tl
2465   \arraybackslash
2466   #5
2467   }
2468   #8
2469   < {
2470   #6

```

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

```

2471   \finalstrut \carstrutbox
2472   \use:c { end #7 }

```

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

```

2473   #4
2474   \@@_cell_end:
2475   }
2476   }
2477 }

```

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

```
2478 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2479 {
```

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 `&`).

```
2480 \group_align_safe_begin:
2481 \peek_meaning:NTF &
2482 { \@@_the_cell_is_empty: }
2483 {
2484     \peek_meaning:NTF \\
2485     { \@@_the_cell_is_empty: }
2486     {
2487         \peek_meaning:NTF \crrc
2488         \@@_the_cell_is_empty:
2489         \group_align_safe_end:
2490     }
2491 }
2492 }
```

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

```
2493 \cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
2494 {
2495     \group_align_safe_begin:
2496     \peek_meaning:NTF &
2497     { \@@_the_cell_is_empty_varwidth: }
2498     {
2499         \peek_meaning:NTF \\
2500         { \@@_the_cell_is_empty_varwidth: }
2501         {
2502             \peek_meaning:NTF \crrc
2503             \@@_the_cell_is_empty_varwidth:
2504             \group_align_safe_end:
2505         }
2506     }
2507 }
2508 \cs_new_protected:Npn \@@_the_cell_is_empty:
2509 {
2510     \group_align_safe_end:
2511     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2512 }
```

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

```
2513     \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).

```
2514     \skip_horizontal:N \l_@@_col_width_dim
2515 }
2516 }
2517 \cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
2518 {
2519     \group_align_safe_end:
2520     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2521     { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2522 }
```

```

2523 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2524 {
2525     \peek_meaning:NT \_siunitx_table_skip:n
2526     { \bool_gset_true:N \g_@@_empty_cell_bool }
2527 }

```

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

```

2528 \cs_new_protected:Npn \@@_center_cell_box:
2529 {

```

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

```

2530 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2531 {
2532     \dim_compare:nNnT
2533     { \box_ht:N \l_@@_cell_box }
2534     >

```

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).

```

2535     { \box_ht:N \strutbox }
2536     {
2537         \hbox_set:Nn \l_@@_cell_box
2538         {
2539             \box_move_down:nn
2540             {
2541                 ( \box_ht:N \l_@@_cell_box - \box_ht:N \arstrutbox
2542                     + \baselineskip ) / 2
2543             }
2544             { \box_use:N \l_@@_cell_box }
2545         }
2546     }
2547 }
2548 }

```

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

```

2549 \cs_new_protected:Npn \@@_V: #1 #2
2550 {
2551     \str_if_eq:nnTF { #2 } { [ ]
2552         { \@@_make_preamble_V_i:w [ ]
2553         { \@@_make_preamble_V_i:w [ ] { #2 } }
2554     }
2555     \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2556     { \@@_make_preamble_V_ii:nn { #1 } }
2557     \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2558     {
2559         \str_set:Nn \l_@@_vpos_col_str { p }
2560         \str_set:Nn \l_@@_hpos_col_str { j }
2561         \@@_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`.

```

2562 \setlength { \l_tmpa_dim } { #2 }
2563 \IfPackageLoadedTF { varwidth }
2564     { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2565     {

```

```

2566     \@@_error_or_warning:n { varwidth-not-loaded }
2567     \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2568   }
2569 }
```

For w and W

```

2570 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2571 \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.
```

```

2572 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2573 {
2574   \str_if_eq:nnTF { #3 } { s }
2575   { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2576   { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2577 }
```

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.
```

```

2578 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2579 {
2580   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2581   \tl_gclear:N \g_@@_pre_cell_tl
2582   \tl_gput_right:Nn \g_@@_array_preamble_tl
2583   {
2584     > {
```

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.

```

2585   \setlength { \l_@@_col_width_dim } { #2 }
2586   \@@_cell_begin:
2587     \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2588   }
2589   c
2590   < {
2591     \@@_cell_end_for_w_s:
2592     #1
2593     \@@_adjust_size_box:
2594     \box_use_drop:N \l_@@_cell_box
2595   }
2596 }
2597 \int_gincr:N \c@jCol
2598 \@@_rec_preamble_after_col:n
2599 }
```

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

```

2600 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2601 {
2602   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2603   \tl_gclear:N \g_@@_pre_cell_tl
2604   \tl_gput_right:Nn \g_@@_array_preamble_tl
2605   {
2606     > {
```

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`.

```

2607      \setlength { \l_@@_col_width_dim } { #4 }
2608      \hbox_set:Nw \l_@@_cell_box
2609      \@@_cell_begin:
2610      \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2611    }
2612    c
2613    < {
2614      \@@_cell_end:
2615      \hbox_set_end:
2616      #1
2617      \@@_adjust_size_box:
2618      \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2619    }
2620  }

```

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

```

2621  \int_gincr:N \c@jCol
2622  \@@_rec_preamble_after_col:n
2623 }

2624 \cs_new_protected:Npn \@@_special_W:
2625 {
2626   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \l_@@_col_width_dim }
2627   { \@@_warning:n { W-warning } }
2628 }

```

For `S` (of `siunitx`).

```

2629 \cs_new_protected:Npn \@@_S: #1 #2
2630 {
2631   \str_if_eq:nnTF { #2 } { [ ]
2632   { \@@_make_preamble_S:w [ ]
2633   { \@@_make_preamble_S:w [ ] { #2 } }
2634 }
2635 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2636 { \@@_make_preamble_S_i:n { #1 } }
2637 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2638 {
2639   \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx-not-loaded } }
2640   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2641   \tl_gclear:N \g_@@_pre_cell_tl
2642   \tl_gput_right:Nn \g_@@_array_preamble_tl
2643   {
2644     > {

```

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).

```

2645   \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2646   \keys_set:mn { siunitx } { #1 }
2647   \@@_cell_begin:
2648   \siunitx_cell_begin:w
2649 }
2650 c
2651 <
2652 {
2653   \siunitx_cell_end:

```

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

```

2654     \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2655     {
2656         \bool_if:NTF \l_siunitx_table_text_bool
2657         { \bool_set_true:N }
2658         { \bool_set_false:N }
2659         \l_siunitx_table_text_bool
2660     }
2661     \@@_cell_end:
2662 }
2663 }
```

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

```

2664     \int_gincr:N \c@jCol
2665     \@@_rec_preamble_after_col:n
2666 }
```

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

```

2667 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2668 {
2669     \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.

```

2670 \int_if_zero:nTF { \c@jCol }
2671 {
2672     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2673 }
```

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

```

2674     \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2675     \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2676     \@@_rec_preamble:n #2
2677 }
2678 {
2679     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2680     \@@_make_preamble_iv:nn { #1 } { #2 }
2681 }
2682 }
2683 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2684 }
2685 \cs_set_eq:cc { @@ _ \token_to_str:N [ : } { @@ _ \token_to_str:N ( : }
2686 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2687 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2688 {
2689     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2690     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2691     \tl_if_in:nnTF { ( [ \{ ] \} ) \left \right } { #2 }
2692     {
2693         \@@_error:nn { delimiter~after~opening } { #2 }
2694         \@@_rec_preamble:n
2695     }
2696     { \@@_rec_preamble:n #2 }
2697 }
```

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

```

2698 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2699 { \use:c { @@ _ \token_to_str:N ( : } }
```

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

```

2700 \cs_new_protected:cpn { @@ _ \token_to_str:N } : } #1 #2
2701 {
2702   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter-with-small } }
2703   \tl_if_in:nnTF { ) ] \} } { #2 }
2704   { \@@_make_preamble_v:nnn #1 #2 }
2705   {
2706     \str_if_eq:nnTF { \s_stop } { #2 }
2707     {
2708       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2709       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2710       {
2711         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2712         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2713         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2714         \@@_rec_preamble:n #2
2715       }
2716     }
2717   {
2718     \tl_if_in:nnT { ( [ \{ \left ] { #2 }
2719     { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2720     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2721     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2722     \@@_rec_preamble:n #2
2723   }
2724 }
2725 }
2726 \cs_set_eq:cc { @@ _ \token_to_str:N } : } { @@ _ \token_to_str:N } : }
2727 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N } : }
2728 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2729 {
2730   \str_if_eq:nnTF { \s_stop } { #3 }
2731   {
2732     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2733     {
2734       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2735       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2736       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2737       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2738     }
2739   {
2740     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2741     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2742     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2743     \@@_error:nn { double-closing-delimiter } { #2 }
2744   }
2745 }
2746 {
2747   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2748   { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2749   \@@_error:nn { double-closing-delimiter } { #2 }
2750   \@@_rec_preamble:n #3
2751 }
2752 }
2753 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2754 { \use:c { @@ _ \token_to_str:N } : } }
```

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

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

```

2755 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2756 {
2757   \str_if_eq:nnTF { #1 } { < }
2758   { \@@_rec_preamble_after_col_i:n }
2759   {
2760     \str_if_eq:nnTF { #1 } { @ }
2761     { \@@_rec_preamble_after_col_ii:n }
2762     {
2763       \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2764       {
2765         \tl_gput_right:Nn \g_@@_array_preamble_tl
2766         { ! { \skip_horizontal:N \arrayrulewidth } }
2767       }
2768       {
2769         \clist_if_in:NeTF \l_@@_vlines_clist
2770         { \int_eval:n { \c@jCol + 1 } }
2771         {
2772           \tl_gput_right:Nn \g_@@_array_preamble_tl
2773           { ! { \skip_horizontal:N \arrayrulewidth } }
2774         }
2775       }
2776     \@@_rec_preamble:n { #1 }
2777   }
2778 }
2779 }

2780 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2781 {
2782   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2783   \@@_rec_preamble_after_col:n
2784 }
```

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

```

2785 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2786 {
2787   \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2788   {
2789     \tl_gput_right:Nn \g_@@_array_preamble_tl
2790     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2791   }
2792   {
2793     \clist_if_in:NeTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2794     {
2795       \tl_gput_right:Nn \g_@@_array_preamble_tl
2796       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2797     }
2798     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2799   }
2800 \@@_rec_preamble:n
2801 }

2802 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2803 {
2804   \tl_clear:N \l_tmpa_tl
2805   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2806   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2807 }
```

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

```

2808 \cs_new_protected:cpn { @@_token_to_str:N \NC@find : } #1
2809 { @@_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.

```

2810 \cs_new_protected:Npn \@@_X: #1 #2
2811 {
2812     \str_if_eq:nnTF { #2 } { [ }
2813     { \@@_make_preamble_X:w [ ] }
2814     { \@@_make_preamble_X:w [ ] #2 }
2815 }
2816 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2817 { \@@_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.

```

2818 \keys_define:nn { nicematrix / X-column }
2819 {
2820     V .code:n =
2821     \IfPackageLoadedTF { varwidth }
2822     {
2823         \bool_set_true:N \l_@@_V_of_X_bool
2824         \bool_gset_true:N \g_@@_V_of_X_bool
2825     }
2826     { \@@_error_or_warning:n { varwidth-not-loaded-in-X } } ,
2827     unknown .code:n =
2828     \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2829     { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2830     { \@@_error_or_warning:n { invalid-weight } }
2831 }

```

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

```

2832 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2833 {

```

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).

```
2834 \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).

```
2835 \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).

```

2836 \fp_set:Nn \l_tmpa_fp { 1.0 }
2837 \@@_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.

```

2838 \bool_set_false:N \l_@@_V_of_X_bool
2839 \keys_set:no { nicematrix / X-column } \l_tmpa_tl

```

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

```
2840 \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).

```
2841 \bool_if:NTF \l_@@_X_columns_aux_bool
2842 {
2843     \@@_make_preamble_ii_iv:nnn
```

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

```
2844 { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2845 { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
2846 { \@@_no_update_width: }
2847 }
```

In the current compilation, we don't known 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.

```
2848 {
2849     \tl_gput_right:Nn \g_@@_array_preamble_tl
2850     {
2851         > {
2852             \@@_cell_begin:
2853             \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`.

```
2854 \NotEmpty
```

The following code will nullify the box of the cell.

```
2855 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2856 { \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`.

```
2857     \begin{minipage}{5cm} \arraybackslash
2858 }
2859     c
2860     < {
2861         \end{minipage}
2862         \@@_cell_end:
2863     }
2864 }
2865 \int_gincr:N \c@jCol
2866 \@@_rec_preamble_after_col:n
2867 }
2868 }

2869 \cs_new_protected:Npn \@@_no_update_width:
2870 {
2871     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2872     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2873 }
```

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

```
2874 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2875 {
2876     \seq_gput_right:Ne \g_@@_cols_vlism_seq
2877     { \int_eval:n { \c@jCol + 1 } }
2878     \tl_gput_right:Ne \g_@@_array_preamble_tl
2879     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2880     \@@_rec_preamble:n
2881 }
```

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.

```
2882 \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).

```
2883 \cs_new_protected:cpn { @@_ \token_to_str:N \hline : }
2884   { @@_fatal:n { Preamble-forgotten } }
2885 \cs_set_eq:cc { @@_ \token_to_str:N \Hline : } { @@_ \token_to_str:N \hline : }
2886 \cs_set_eq:cc { @@_ \token_to_str:N \toprule : }
2887   { @@_ \token_to_str:N \hline : }
2888 \cs_set_eq:cc { @@_ \token_to_str:N \Block : } { @@_ \token_to_str:N \hline : }
2889 \cs_set_eq:cc { @@_ \token_to_str:N \CodeBefore : }
2890   { @@_ \token_to_str:N \hline : }
2891 \cs_set_eq:cc { @@_ \token_to_str:N \RowStyle : }
2892   { @@_ \token_to_str:N \hline : }
2893 \cs_set_eq:cc { @@_ \token_to_str:N \diagbox : }
2894   { @@_ \token_to_str:N \hline : }
2895 \cs_set_eq:cc { @@_ \token_to_str:N & : }
2896   { @@_ \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).

```
2897 \cs_new:Npn @@_multicolumn:nnn #1 #2 #
2898 {
```

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`.

```
2899 \multispan { #1 }
2900 \cs_set_eq:NN @@_update_max_cell_width: \prg_do_nothing:
2901 \begingroup
2902 \tbl_update_multicolumn_cell_data:n { #1 }
```

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

```
2903 \tl_gclear:N \g_@@_preamble_tl
2904 \@@_make_m_preamble:n #2 \q_stop
```

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

```
2905 \def \@addamp
2906 {
2907   \legacy_if:nTF { @firstamp }
2908   { \legacy_if_set_false:n { @firstamp } }
2909   { \preamerr 5 }
2910 }
2911 \exp_args:No \omkpream \g_@@_preamble_tl
2912 \@addtopreamble \empty
2913 \endgroup
2914 \UseTaggingSocket { tbl / colspan } { #1 }
```

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

```
2915 \int_compare:nNnT { #1 } > { \c_one_int }
2916 {
2917   \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2918   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2919   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
```

```

2920 \seq_gput_right:Nn \g_@@_pos_of_blocks_seq
2921 {
2922   {
2923     \int_if_zero:nTF { \c@jCol }
2924       { \int_eval:n { \c@iRow + 1 } }
2925       { \int_use:N \c@iRow }
2926   }
2927   { \int_eval:n { \c@jCol + 1 } }
2928   {
2929     \int_if_zero:nTF { \c@jCol }
2930       { \int_eval:n { \c@iRow + 1 } }
2931       { \int_use:N \c@iRow }
2932   }
2933   { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block.

```

2934   { }
2935   }
2936 }
```

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

```

2937 \RenewDocumentCommand { \cellcolor } { O{ } m }
2938 {
2939   \tl_gput_right:Nn \g_@@_pre_code_before_tl
2940   {
2941     \@@_rectanglecolor [ ##1 ]
2942       { \exp_not:n { ##2 } }
2943       { \int_use:N \c@iRow - \int_use:N \c@jCol }
2944       { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2945   }
2946   \ignorespaces
2947 }
```

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

```

2948 \def \sharp { #3 }
2949 \carstrut
2950 \preamble
2951 \null
```

We add some lines.

```

2952 \int_gadd:Nn \c@jCol { #1 - 1 }
2953 \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
2954   { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2955 \ignorespaces
2956 }
```

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`.

```

2957 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2958 {
2959   \str_case:nnF { #1 }
2960   {
2961     c { \@@_make_m_preamble_i:n #1 }
2962     l { \@@_make_m_preamble_i:n #1 }
2963     r { \@@_make_m_preamble_i:n #1 }
2964     > { \@@_make_m_preamble_ii:nn #1 }
2965     ! { \@@_make_m_preamble_ii:nn #1 }
2966     @ { \@@_make_m_preamble_ii:nn #1 }
2967     | { \@@_make_m_preamble_iii:n #1 }
2968     p { \@@_make_m_preamble_iv:nnn t #1 }
2969     m { \@@_make_m_preamble_iv:nnn c #1 }

```

```

2970     b { \@@_make_m_preamble_iv:nnn b #1 }
2971     w { \@@_make_m_preamble_v:nnnn { } #1 }
2972     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2973     \q_stop { }
2974   }
2975   {
2976     \cs_if_exist:cTF { NC @ find @ #1 }
2977     {
2978       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2979       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2980     }
2981     {
2982       \str_if_eq:nnTF { #1 } { S }
2983         { \@@_fatal:n { unknown~column~type~S~multicolumn } }
2984         { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
2985     }
2986   }
2987 }
```

For c, l and r

```

2988 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2989   {
2990     \tl_gput_right:Nn \g_@@_preamble_tl
2991     {
2992       > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
2993       #1
2994       < \@@_cell_end:
2995     }
2996 }
```

We test for the presence of a <.

```

2996   \@@_make_m_preamble_x:n
2997 }
```

For >, ! and @

```

2998 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2999   {
3000     \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
3001     \@@_make_m_preamble:n
3002 }
```

For |

```

3003 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
3004   {
3005     \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
3006     \@@_make_m_preamble:n
3007 }
```

For p, m and b

```

3008 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
3009   {
3010     \tl_gput_right:Nn \g_@@_preamble_tl
3011     {
3012       > {
3013         \@@_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`.

```

3014   \setlength { \l_tmpa_dim } { #3 }
3015   \begin { minipage } [ #1 ] { \l_tmpa_dim }
3016   \mode_leave_vertical:
3017   \arraybackslash
3018   \vrule height \box_ht:N \carstrutbox depth \c_zero_dim width \c_zero_dim
```

```

3019         }
3020         c
3021         < {
3022             \vrule height \c_zero_dim depth \box_dp:N \carstrutbox width \c_zero_dim
3023             \end { minipage }
3024             \@@_cell_end:
3025         }
3026     }

```

We test for the presence of a <.

```

3027     \@@_make_m_preamble_x:n
3028 }

```

For w and W

```

3029 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3030 {
3031     \tl_gput_right:Nn \g_@@_preamble_tl
3032     {
3033         > {
3034             \dim_set:Nn \l_@@_col_width_dim { #4 }
3035             \hbox_set:Nw \l_@@_cell_box
3036             \@@_cell_begin:
3037             \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3038         }
3039         c
3040         < {
3041             \@@_cell_end:
3042             \hbox_set_end:
3043             \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3044             #1
3045             \@@_adjust_size_box:
3046             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3047         }
3048     }

```

We test for the presence of a <.

```

3049     \@@_make_m_preamble_x:n
3050 }

```

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

```

3051 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3052 {
3053     \str_if_eq:nnTF { #1 } { < }
3054     { \@@_make_m_preamble_ix:n }
3055     { \@@_make_m_preamble:n { #1 } }
3056 }
3057 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3058 {
3059     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3060     \@@_make_m_preamble_x:n
3061 }

```

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`).

```

3062 \cs_new_protected:Npn \@@_put_box_in_flow:
3063 {
3064     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3065     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3066     \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
3067     { \box_use_drop:N \l_tmpa_box }
3068     { \@@_put_box_in_flow_i: }
3069 }

```

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

```
3070 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3071 {
3072     \pgfpicture
3073         \@@_qpoint:n { row - 1 }
3074         \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3075         \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3076         \dim_gadd:Nn \g_tmpa_dim \pgf@y
3077         \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).

```
3078     \tl_if_in:NnTF \l_@baseline_tl { line- }
3079     {
3080         \int_set:Nn \l_tmpa_int
3081             { \str_range:Nnn \l_@baseline_tl { 6 } { -1 } }
3082         \bool_lazy_or:nnT
3083             { \int_compare_p:nNn { \l_tmpa_int } < { 1 } }
3084             { \int_compare_p:nNn { \l_tmpa_int } > { \c@iRow + 1 } }
3085             {
3086                 \@@_error:n { bad-value-for-baseline-line }
3087                 \int_set_eq:NN \l_tmpa_int \c_one_int
3088             }
3089         \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3090     }
3091     {
3092         \str_if_eq:eeTF { \l_@baseline_tl } { t }
3093             { \int_set_eq:NN \l_tmpa_int \c_one_int }
3094             {
3095                 \str_if_eq:onTF \l_@baseline_tl { b }
3096                     { \int_set_eq:NN \l_tmpa_int \c@iRow }
3097                     { \int_set:Nn \l_tmpa_int \l_@baseline_tl }
3098             }
3099         \bool_lazy_or:nnT
3100             { \int_compare_p:nNn { \l_tmpa_int } < { \l_@first_row_int } }
3101             { \int_compare_p:nNn { \l_tmpa_int } > { \g_@row_total_int } }
3102             {
3103                 \@@_error:n { bad-value-for-baseline }
3104                 \int_set_eq:NN \l_tmpa_int \c_one_int
3105             }
3106         \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
```

We take into account the position of the mathematical axis.

```
3107     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3108     }
3109     \dim_gsub:Nn \g_tmpa_dim \pgf@y
```

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

```
3110     \endpgfpicture
3111     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3112     \box_use_drop:N \l_tmpa_box
3113 }
```

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).

```
3114 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3115 {
```

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.

```
3116     \bool_if:NT \l_@NiceMatrix_without_vlines_bool
```

```

3117    {
3118        \int_compare:nNnT { \c@jCol } > { \c_one_int }
3119        {
3120            \box_set_wd:Nn \l_@@_the_array_box
3121            { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3122        }
3123    }

```

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

```

3124    \begin{minipage} [ t ] { \box_wd:N \l_@@_the_array_box }
3125        \bool_if:NT \l_@@_caption_above_bool
3126        {
3127            \tl_if_empty:NF \l_@@_caption_tl
3128            {
3129                \bool_set_false:N \g_@@_caption_finished_bool
3130                \int_gzero:N \c@tabularnote
3131                \@@_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).

```

3132    \int_compare:nNnT { \g_@@_notes_caption_int } > { \c_zero_int }
3133    {
3134        \tl_gput_right:Ne \g_@@_aux_tl
3135        {
3136            \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3137            { \int_use:N \g_@@_notes_caption_int }
3138        }
3139        \int_gzero:N \g_@@_notes_caption_int
3140    }
3141}
3142}

```

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

```

3143 \hbox
3144 {
3145     \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.

```

3146     \@@_create_extra_nodes:
3147     \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3148 }

```

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).

```

3149 \bool_lazy_any:nT
3150 {
3151     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3152     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3153     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3154 }
3155 \@@_insert_tabularnotes:
3156 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3157 \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3158 \end{minipage}
3159 }

```

```

3160 \cs_new_protected:Npn \@@_insert_caption:
3161 {
3162     \tl_if_empty:NF \l_@@_caption_tl
3163     {
3164         \cs_if_exist:NTF \c@ptyp
3165         { \@@_insert_caption_i: }
3166         { \@@_error:n { caption-outside-float } }
3167     }
3168 }

3169 \cs_new_protected:Npn \@@_insert_caption_i:
3170 {
3171     \group_begin:

```

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

```
3172     \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.

```

3173     \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \makecaption \FR@makecaption }
3174     \tl_if_empty:NTF \l_@@_short_caption_tl
3175     {
3176         \caption
3177         { \caption [ \l_@@_short_caption_tl ] }
3178         { \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.

```

3178     \bool_if:NF \g_@@_caption_finished_bool
3179     {
3180         \bool_gset_true:N \g_@@_caption_finished_bool
3181         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3182         \int_gzero:N \c@tabularnote
3183     }
3184     \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3185     \group_end:
3186 }

3187 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3188 {
3189     \@@_error_or_warning:n { tabularnote-below-the-tabular }
3190     \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3191 }

3192 \cs_new_protected:Npn \@@_insert_tabularnotes:
3193 {
3194     \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3195     \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3196     \skip_vertical:N 0.65ex

```

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

```

3197 \group_begin:
3198 \l_@@_notes_code_before_tl
3199 \tl_if_empty:NF \g_@@_tabularnote_tl
3200 {
3201     \g_@@_tabularnote_tl \par
3202     \tl_gclear:N \g_@@_tabularnote_tl
3203 }

```

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.

```

3204   \int_compare:nNnT { \c@tabularnote } > { \c_zero_int }
3205   {
3206     \bool_if:NTF \l_@@_notes_para_bool
3207     {
3208       \begin { tabularnotes* }
3209         \seq_map_inline:Nn \g_@@_notes_seq
3210           { \@@_one_tabularnote:nn ##1 }
3211         \strut
3212       \end { tabularnotes* }

```

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

```

3213   \par
3214   }
3215   {
3216     \tabularnotes
3217       \seq_map_inline:Nn \g_@@_notes_seq
3218         { \@@_one_tabularnote:nn ##1 }
3219         \strut
3220       \endtabularnotes
3221     }
3222   }
3223   \unskip
3224   \group_end:
3225   \bool_if:NT \l_@@_notes_bottomrule_bool
3226   {
3227     \IfPackageLoadedTF { booktabs }
3228     {

```

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

```

3229   \skip_vertical:N \aboverulesep

```

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

```

3230   { \CT@arc@ \hrule height \heavyrulewidth }
3231   }
3232   { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3233   }
3234   \l_@@_notes_code_after_tl
3235   \seq_gclear:N \g_@@_notes_seq
3236   \seq_gclear:N \g_@@_notes_in_caption_seq
3237   \int_gzero:N \c@tabularnote
3238 }

```

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

```

3239 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3240   {
3241     \tl_if_no_value:nTF { #1 }
3242       { \item }
3243       { \item [ \@@_notes_label_in_list:n { #1 } ] }
3244   }

```

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`.

```

3245 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3246   {
3247     \pgfpicture
3248       \@@_qpoint:n { row - 1 }
3249       \dim_gset_eq:NN \g_tmpa_dim \pgf@y

```

```

3250 \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3251   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3252 \endpgfpicture
3253 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3254 \int_if_zero:nT { \l_@@_first_row_int }
3255 {
3256   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3257   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3258 }
3259 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3260 }

```

Now, the general case.

```

3261 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3262 {

```

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

```

3263 \str_if_eq:eeT { \l_@@_baseline_tl } { t }
3264   { \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 .

```

3265 \pgfpicture
3266 \@@_qpoint:n { row - 1 }
3267 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3268 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3269 {
3270   \int_set:Nn \l_tmpa_int
3271   {
3272     \str_range:Nnn
3273       \l_@@_baseline_tl
3274       { 6 }
3275       { \tl_count:o \l_@@_baseline_tl }
3276   }
3277 \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3278 }
3279 {
3280   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3281   \bool_lazy_or:mnT
3282   {
3283     { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3284     { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3285   }
3286   \@@_error:n { bad-value-for-baseline }
3287   \int_set:Nn \l_tmpa_int 1
3288   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3289 }
3290 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3291 \endpgfpicture
3292 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3293 \int_if_zero:nT { \l_@@_first_row_int }
3294 {
3295   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3296   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3297 }
3298 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3299 }

```

The command $\text{\@@_put_box_in_flow_bis:}$ 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.

```

3300 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3301 {

```

We will compute the real width of both delimiters used.

```

3302 \dim_zero_new:N \l_@@_real_left_delim_dim
3303 \dim_zero_new:N \l_@@_real_right_delim_dim
3304 \hbox_set:Nn \l_tmpb_box
3305 {
3306     \m@th
3307     $ % $
3308     \left #1
3309     \vcenter
3310     {
3311         \vbox_to_ht:nn
3312         { \box_ht_plus_dp:N \l_tmpa_box }
3313         { }
3314     }
3315     \right .
3316     $ % $
3317 }
3318 \dim_set:Nn \l_@@_real_left_delim_dim
3319 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3320 \hbox_set:Nn \l_tmpb_box
3321 {
3322     \m@th
3323     $ % $
3324     \left .
3325     \vbox_to_ht:nn
3326     { \box_ht_plus_dp:N \l_tmpa_box }
3327     { }
3328     \right #2
3329     $ % $
3330 }
3331 \dim_set:Nn \l_@@_real_right_delim_dim
3332 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3333 \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3334 \@@_put_box_in_flow:
3335 \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3336 }

```

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).

```
3337 \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).

```

3338 {
3339     \peek_remove_spaces:n
3340     {
3341         \peek_meaning:NTF \end
3342         { \@@_analyze_end:Nn }
3343         {
3344             \@@_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`).

```

3345     \@@_array:o \g_@@_array_preamble_tl
3346     }
3347 }
3348 }
```

```

3349 {
3350   \@@_create_col_nodes:
3351   \endarray
3352 }

```

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

```

3353 \NewDocumentEnvironment { @@-light-syntax } { b }
3354 {

```

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`.

```

3355 \tl_if_empty:nT { #1 }
3356   { \@@_fatal:n { empty~environment } }
3357 \tl_if_in:nnT { #1 } { & }
3358   { \@@_fatal:n { ampersand~in~light~syntax } }
3359 \tl_if_in:nnT { #1 } { \\ }
3360   { \@@_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`.

```
3361   \@@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```
3362 }
```

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.

```

3363 {
3364   \@@_create_col_nodes:
3365   \endarray
3366 }

3367 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3368 {
3369   \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).

```
3370   \seq_clear_new:N \l_@@_rows_seq
```

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

```

3371 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3372 \bool_if:NTF \l_@@_light_syntax_expanded_bool
3373   { \seq_set_split:Nee }
3374   { \seq_set_split:Non }
3375   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3376 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3377 \tl_if_empty:NF \l_tmpa_tl
3378   { \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.

```

3379 \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
3380   { \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`.

```
3381 \tl_build_begin:N \l_@@_new_body_tl
3382 \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3383 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3384 \@@_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).

```
3385 \seq_map_inline:Nn \l_@@_rows_seq
3386 {
3387     \tl_build_put_right:Nn \l_@@_new_body_tl { \& }
3388     \@@_line_with_light_syntax:n { ##1 }
3389 }
3390 \tl_build_end:N \l_@@_new_body_tl
3391 \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
3392 {
3393     \int_set:Nn \l_@@_last_col_int
3394     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3395 }
```

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.

```
3396 \@@_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`).

```
3397 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3398 }
3399 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3400 {
3401     \seq_clear_new:N \l_@@_cells_seq
3402     \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3403     \int_set:Nn \l_@@_nb_cols_int
3404     {
3405         \int_max:nn
3406         { \l_@@_nb_cols_int }
3407         { \seq_count:N \l_@@_cells_seq }
3408     }
3409     \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3410     \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3411     \seq_map_inline:Nn \l_@@_cells_seq
3412     { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3413 }
3414 \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`.

```
3415 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3416 {
3417     \str_if_eq:eeT { \g_@@_name_env_str } { #2 }
3418     { \@@_fatal:n { empty-environment } }
```

We reput in the stream the `\end{...}` we have extracted and the user will have an error for incorrect nested environments.

```
3419     \end { #2 }
3420 }
```

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`).

```

3421 \cs_new:Npn \@@_create_col_nodes:
3422 {
3423     \crr
3424     \int_if_zero:nT { \l_@@_first_col_int }
3425     {
3426         \omit
3427         \hbox_overlap_left:n
3428         {
3429             \bool_if:NT \l_@@_code_before_bool
3430                 { \pgfsys@markposition { \@@_env: - col - 0 } }
3431             \pgfpicture
3432             \pgfrememberpicturepositiononpagetrue
3433             \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3434             \str_if_empty:NF \l_@@_name_str
3435                 { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3436             \endpgfpicture
3437             \skip_horizontal:n { 2 \colsep + \g_@@_width_first_col_dim }
3438         }
3439         &
3440     }
3441     \omit

```

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

```
3442     \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`).

```

3443     \int_if_zero:nTF { \l_@@_first_col_int }
3444     {
3445         \@@_mark_position:n { 1 }
3446         \pgfpicture
3447         \pgfrememberpicturepositiononpagetrue
3448         \pgfcoordinate { \@@_env: - col - 1 }
3449             { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3450         \str_if_empty:NF \l_@@_name_str
3451             { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3452         \endpgfpicture
3453     }
3454     {
3455         \bool_if:NT \l_@@_code_before_bool
3456             {
3457                 \hbox
3458                 {
3459                     \skip_horizontal:n { 0.5 \arrayrulewidth }
3460                     \pgfsys@markposition { \@@_env: - col - 1 }
3461                     \skip_horizontal:n { -0.5 \arrayrulewidth }
3462                 }
3463             }
3464         \pgfpicture
3465         \pgfrememberpicturepositiononpagetrue
3466         \pgfcoordinate { \@@_env: - col - 1 }
3467             { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3468         \@@_node_alias:n { 1 }
3469         \endpgfpicture
3470     }

```

We compute in `\g_tma_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_tma_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.

```

3471  \skip_gset:Nn \g_tmpa_skip { 0 pt plus 1 fill }
3472  \bool_if:NF \l_@@_auto_columns_width_bool
3473  { \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim } }
3474  {
3475    \bool_lazy_and:nnTF
3476    { \l_@@_auto_columns_width_bool }
3477    { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3478    { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3479    { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3480    \skip_gadd:Nn \g_tmpa_skip { 2 \colsep }
3481  }
3482  \skip_horizontal:N \g_tmpa_skip
3483  \hbox
3484  {
3485    \@@_mark_position:n { 2 }
3486    \pgfpicture
3487    \pgfrememberpicturepositiononpagetrue
3488    \pgfcoordinate { \@@_env: - col - 2 }
3489    { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3490    \@@_node_alias:n { 2 }
3491    \endpgfpicture
3492  }

```

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.

```

3493  \int_gset_eq:NN \g_tmpa_int \c_one_int
3494  \bool_if:NTF \g_@@_last_col_found_bool
3495  { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3496  { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3497  {
3498    &
3499    \omit
3500    \int_gincr:N \g_tmpa_int

```

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

```

3501  \skip_horizontal:N \g_tmpa_skip
3502  \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

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

```

3503  \pgfpicture
3504    \pgfrememberpicturepositiononpagetrue
3505    \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3506    { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3507    \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3508    \endpgfpicture
3509
3510    % &      % moved on 2025-11-02
3511    % \omit

```

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

```

3512  \bool_lazy_or:nnF
3513  { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3514  { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3515  {
3516    &
3517    \omit
3518    \skip_horizontal:N \g_tmpa_skip
3519    \int_gincr:N \g_tmpa_int
3520    \bool_lazy_any:nF
3521    {

```

```

3522     \g_@@_delims_bool
3523     \l_@@_tabular_bool
3524     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3525     \l_@@_exterior_arraycolsep_bool
3526     \l_@@_bar_at_end_of_pream_bool
3527   }
3528   { \skip_horizontal:n { - \col@sep } }
3529 \bool_if:NT \l_@@_code_before_bool
3530   {
3531     \hbox
3532   {
3533     \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.

```

3534     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3535       { \skip_horizontal:n { - \arraycolsep } }
3536     \pgfsys@markposition
3537       { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3538       \skip_horizontal:n { 0.5 \arrayrulewidth }
3539     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3540       { \skip_horizontal:N \arraycolsep }
3541   }
3542 }
3543 \pgfpicture
3544   \pgfrememberpicturepositiononpagetrue
3545   \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3546   {
3547     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3548   {
3549     \pgfpoint
3550       { - 0.5 \arrayrulewidth - \arraycolsep }
3551       \c_zero_dim
3552   }
3553   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3554 }
3555 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3556 \endpgfpicture
3557 }

3558 \bool_if:NT \g_@@_last_col_found_bool
3559   {
3560     \hbox_overlap_right:n
3561   {
3562     \skip_horizontal:N \g_@@_width_last_col_dim
3563     \skip_horizontal:N \col@sep
3564     \bool_if:NT \l_@@_code_before_bool
3565   {
3566     \pgfsys@markposition
3567       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3568   }
3569   \pgfpicture
3570   \pgfrememberpicturepositiononpagetrue
3571   \pgfcoordinate
3572     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3573     \pgfpointorigin
3574   \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3575   \endpgfpicture
3576 }
3577 }
3578

```

```

3579 \cs_new_protected:Npn \@@_mark_position:n #1
3580 {
3581     \bool_if:NT \l_@@_code_before_bool
3582     {
3583         \hbox
3584         {
3585             \skip_horizontal:n { -0.5 \arrayrulewidth }
3586             \pgfsys@markposition { \@@_env: - col - #1 }
3587             \skip_horizontal:n { 0.5 \arrayrulewidth }
3588         }
3589     }
3590 }
3591 \cs_new_protected:Npn \@@_node_alias:n #1
3592 {
3593     \str_if_empty:NF \l_@@_name_str
3594     { \pgfnodealias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3595 }

```

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

```

3596 \tl_const:Nn \c_@@_preamble_first_col_tl
3597 {
3598     >
3599 }

```

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

```

3600 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3601 \bool_gset_true:N \g_@@_after_col_zero_bool
3602 \@@_begin_of_row:
3603 \hbox_set:Nw \l_@@_cell_box
3604 \@@_math_toggle:
3605 \@@_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”.

```

3606 \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3607 {
3608     \bool_lazy_or:nnT
3609     { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3610     { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3611     {
3612         \l_@@_code_for_first_col_tl
3613         \xglobal \colorlet { nicematrix-first-col } { . }
3614     }
3615 }
3616 }

```

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`.

```

3617 l
3618 <
3619 {
3620     \@@_math_toggle:
3621     \hbox_set_end:
3622     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3623     \@@_adjust_size_box:
3624     \@@_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.

```

3625 \dim_gset:Nn \g_@@_width_first_col_dim
3626     { \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.

```

3627 \hbox_overlap_left:n
3628 {
3629     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3630         { \c_@@_node_cell: }
3631         { \box_use_drop:N \l_@@_cell_box }
3632         \skip_horizontal:N \l_@@_left_delim_dim
3633         \skip_horizontal:N \l_@@_left_margin_dim
3634         \skip_horizontal:N \l_@@_extra_left_margin_dim
3635     }
3636     \bool_gset_false:N \g_@@_empty_cell_bool
3637     \skip_horizontal:n { -2 \col@sep }
3638 }
3639 }
```

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

```

3640 \tl_const:Nn \c_@@_preamble_last_col_tl
3641 {
3642     >
3643     {
3644         \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).

```
3645 \cs_set_eq:NN \CodeAfter \c_@@_CodeAfter_i:
```

With the flag `\g_@@_last_col_found_bool`, we will know that the “last column” is really used.

```

3646 \bool_gset_true:N \g_@@_last_col_found_bool
3647 \int_gincr:N \c@jCol
3648 \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3649 \hbox_set:Nw \l_@@_cell_box
3650     \c@_math_toggle:
3651     \c@_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”.

```

3652 \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3653 {
3654     \bool_lazy_or:nnT
3655         { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3656         { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3657     {
3658         \l_@@_code_for_last_col_tl
3659         \xglobal \colorlet { nicematrix-last-col } { . }
3660     }
3661 }
3662 }
3663 1
3664 <
3665 {
3666     \c@_math_toggle:
3667     \hbox_set_end:
3668     \bool_if:NT \g_@@_rotate_bool { \c@_rotate_cell_box: }
3669     \c@_adjust_size_box:
3670     \c@_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.

```

3671 \dim_gset:Nn \g_@@_width_last_col_dim
3672     { \dim_max:nn { \g_@@_width_last_col_dim } { \box_wd:N \l_@@_cell_box } }
3673     \skip_horizontal:n { -2 \col@sep }
```

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

```

3674 \hbox_overlap_right:n
3675 {
```

```

3676     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3677     {
3678         \skip_horizontal:N \l_@@_right_delim_dim
3679         \skip_horizontal:N \l_@@_right_margin_dim
3680         \skip_horizontal:N \l_@@_extra_right_margin_dim
3681         \node_cell:
3682     }
3683 }
3684 \bool_gset_false:N \g_@@_empty_cell_bool
3685 }
3686 }
```

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

```

3687 \NewDocumentEnvironment { NiceArray } { }
3688 {
3689     \bool_gset_false:N \g_@@_delims_bool
3690     \str_if_empty:NT \g_@@_name_env_str
3691     { \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).

```

3692     \NiceArrayWithDelims . .
3693 }
3694 { \endNiceArrayWithDelims }
```

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

```

3695 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3696 {
3697     \NewDocumentEnvironment { #1 NiceArray } { }
3698     {
3699         \bool_gset_true:N \g_@@_delims_bool
3700         \str_if_empty:NT \g_@@_name_env_str
3701         { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3702         \@@_test_if_math_mode:
3703         \NiceArrayWithDelims #2 #3
3704     }
3705     { \endNiceArrayWithDelims }
3706 }
```

3707 \@@_def_env:NNN p ()
3708 \@@_def_env:NNN b []
3709 \@@_def_env:NNN B \{ \}
3710 \@@_def_env:NNN v \vert \vert
3711 \@@_def_env:NNN V \Vert \Vert

13 The environment `{NiceMatrix}` and its variants

```

3712 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3713 {
3714     \bool_set_false:N \l_@@_preamble_bool
3715     \tl_clear:N \l_tmpa_tl
3716     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3717     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3718     \tl_put_right:Nn \l_tmpa_tl
3719     {
3720         *
3721     }
```

```

3722     \int_case:nnF \l_@@_last_col_int
3723     {
3724         { -2 } { \c@MaxMatrixCols }
3725         { -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).

```

3726     }
3727     { \int_eval:n { \l_@@_last_col_int - 1 } }
3728 }
3729 { #2 }
3730 }
3731 \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3732 \exp_args:No \l_tmpb_tl \l_tmpa_tl
3733 }
3734 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3735 \clist_map_inline:nn { p , b , B , v , V }
3736 {
3737     \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3738     {
3739         \bool_gset_true:N \g_@@_delims_bool
3740         \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3741         \int_if_zero:nT { \l_@@_last_col_int }
3742         {
3743             \bool_set_true:N \l_@@_last_col_without_value_bool
3744             \int_set:Nn \l_@@_last_col_int { -1 }
3745         }
3746         \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3747         \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3748     }
3749     { \use:c { end #1 NiceArray } }
3750 }

```

We define also an environment {NiceMatrix}

```

3751 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3752 {
3753     \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3754     \int_if_zero:nT { \l_@@_last_col_int }
3755     {
3756         \bool_set_true:N \l_@@_last_col_without_value_bool
3757         \int_set:Nn \l_@@_last_col_int { -1 }
3758     }
3759     \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3760     \bool_lazy_or:nnT
3761     { \clist_if_empty_p:N \l_@@_vlines_clist }
3762     { \l_@@_except_borders_bool }
3763     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3764     \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3765 }
3766 { \endNiceArray }

```

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

```

3767 \cs_new_protected:Npn \@@_NotEmpty:
3768     { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3769 \NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
3770 {

```

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.

```

3771 \dim_compare:nNnT { \l_@@_width_dim } = { \c_zero_dim }
3772   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3773 \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3774 \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3775 \tl_if_empty:NF \l_@@_short_caption_tl
3776 {
3777   \tl_if_empty:NT \l_@@_caption_tl
3778   {
3779     \@@_error_or_warning:n { short-caption-without-caption }
3780     \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3781   }
3782 }
3783 \tl_if_empty:NF \l_@@_label_tl
3784 {
3785   \tl_if_empty:NT \l_@@_caption_tl
3786   { \@@_error_or_warning:n { label-without-caption } }
3787 }
3788 \NewDocumentEnvironment { TabularNote } { b }
3789 {
3790   \bool_if:NTF \l_@@_in_code_after_bool
3791   { \@@_error_or_warning:n { TabularNote-in-CodeAfter } }
3792   {
3793     \tl_if_empty:NF \g_@@_tabularnote_tl
3794     { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3795     \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3796   }
3797 }
3798 {
3799 \@@_settings_for_tabular:
3800 \NiceArray { #2 }
3801 }
3802 { \endNiceArray }
3803 \cs_new_protected:Npn \@@_settings_for_tabular:
3804 {
3805   \bool_set_true:N \l_@@_tabular_bool
3806   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3807   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3808   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3809 }

3810 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3811 {
3812   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3813   \dim_set:Nn \l_@@_width_dim { #1 }
3814   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3815 \@@_settings_for_tabular:
3816 \NiceArray { #3 }
3817 }
3818 {
3819 \endNiceArray
3820 \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3821   { \@@_error:n { NiceTabularX-without-X } }
3822 }

3823 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3824 {
3825   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3826   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3827   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3828 \@@_settings_for_tabular:
3829 \NiceArray { #3 }
3830 }
3831 { \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`).

```

3832 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3833 {
3834     \bool_lazy_all:nT
3835     {
3836         { \dim_compare_p:nNn { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim } }
3837         { \l_@@_hvlines_bool }
3838         { ! \g_@@_delims_bool }
3839         { ! \l_@@_except_borders_bool }
3840     }
3841     {
3842         \bool_set_true:N \l_@@_except_borders_bool
3843         \clist_if_empty:NF \l_@@_corners_clist
3844             { \@@_error:n { hvlines,~rounded-corners-and-corners } }
3845         \tl_gput_right:Nn \g_@@_pre_code_after_tl
3846             {
3847                 \@@_stroke_block:nnn
3848                 {
3849                     rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3850                     draw = \l_@@_rules_color_tl
3851                 }
3852                 { 1-1 }
3853                 { \int_use:N \c@iRow - \int_use:N \c@jCol }
3854             }
3855     }
3856 }
3857 \cs_new_protected:Npn \@@_after_array:
3858 {

```

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`.

```

3859     \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3860     \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.

```

3861     \bool_if:NT \g_@@_last_col_found_bool
3862         { \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`.

```

3863     \bool_if:NT \l_@@_last_col_without_value_bool
3864         { \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.

```

3865     \bool_if:NT \l_@@_last_row_without_value_bool
3866         { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3867   \tl_gput_right:Ne \g_@@_aux_tl
3868   {
3869     \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3870     {
3871       \int_use:N \l_@@_first_row_int ,
3872       \int_use:N \c@iRow ,
3873       \int_use:N \g_@@_row_total_int ,
3874       \int_use:N \l_@@_first_col_int ,
3875       \int_use:N \c@jCol ,
3876       \int_use:N \g_@@_col_total_int
3877     }
3878   }

```

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`.

```

3879   \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3880   {
3881     \tl_gput_right:Ne \g_@@_aux_tl
3882     {
3883       \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3884       { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }
3885     }
3886   }
3887   \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3888   {
3889     \tl_gput_right:Ne \g_@@_aux_tl
3890     {
3891       \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3892       { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
3893       \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3894       { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
3895     }
3896   }

```

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

```
3897   \@@_create_diag_nodes:
```

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

```

3898   \pgfpicture
3899   \@@_create_aliases_last:
3900   \str_if_empty:NF \l_@@_name_str { \@@_create_alias_nodes: }
3901   \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.

```

3902   \bool_if:NT \l_@@_parallelize_diags_bool
3903   {
3904     \int_gzero:N \g_@@_ddots_int
3905     \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.

```

3906     \dim_gzero:N \g_@@_delta_x_one_dim
3907     \dim_gzero:N \g_@@_delta_y_one_dim
3908     \dim_gzero:N \g_@@_delta_x_two_dim
3909     \dim_gzero:N \g_@@_delta_y_two_dim
3910   }

```

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

```

3911 \bool_set_false:N \l_@@_initial_open_bool
3912 \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.

```

3913 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

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

```

3914 \@@_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.

```

3915 \clist_if_empty:NF \l_@@_corners_clist
3916 {
3917     \bool_if:NTF \l_@@_no_cell_nodes_bool
3918         { \@@_error:n { corners~with~no~cell~nodes } }
3919         { \@@_compute_corners: }
3920 }

```

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`.

```

3921 \seq_gconcat:NNN \g_@@_pos_of_blocks_seq
3922     \g_@@_pos_of_blocks_seq
3923     \g_@@_future_pos_of_blocks_seq
3924 \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-*}`).

```

3925 \@@_adjust_pos_of_blocks_seq:
3926 \@@_deal_with_rounded_corners:
3927 \clist_if_empty:NF \l_@@_hlines_clist { \@@_draw_hlines: }
3928 \clist_if_empty:NF \l_@@_vlines_clist { \@@_draw_vlines: }

```

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

```

3929 \IfPackageLoadedT { tikz }
3930 {
3931     \tikzset
3932     {
3933         every~picture / .style =
3934         {
3935             overlay ,
3936             remember~picture ,
3937             name~prefix = \@@_env: -
3938         }
3939     }
3940 }
3941 \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
3942 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3943 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3944 \cs_set_eq:NN \OverBrace \@@_OverBrace
3945 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3946 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3947 \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.

```

3948 \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
3949 \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.

```
3950     \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).

```
3951     \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`).

```
3952     \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3953         { \@@_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`:

```
3954     \bool_set_true:N \l_@@_in_code_after_bool
3955     \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3956     \scan_stop:
3957     \tl_gclear:N \g_nicematrix_code_after_tl
3958     \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.

```
3959     \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3960     \tl_if_empty:NF \g_@@_pre_code_before_tl
3961     {
3962         \tl_gput_right:Ne \g_@@_aux_tl
3963         {
3964             \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3965             { \exp_not:o \g_@@_pre_code_before_tl }
3966         }
3967         \tl_gclear:N \g_@@_pre_code_before_tl
3968     }
3969     \tl_if_empty:NF \g_nicematrix_code_before_tl
3970     {
3971         \tl_gput_right:Ne \g_@@_aux_tl
3972         {
3973             \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3974             { \exp_not:o \g_nicematrix_code_before_tl }
3975         }
3976         \tl_gclear:N \g_nicematrix_code_before_tl
3977     }

3978     \str_gclear:N \g_@@_name_env_str
3979     \@@_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`.

```
3980     \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3981 }
```

¹³e.g. `\color[rgb]{0.5,0.5,0}`

```

3982 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
3983 {
3984     \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3985     \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }

```

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

```

3986     \dim_set:Nn \l_@@_xdots_shorten_start_dim
3987         { 0.6 \l_@@_xdots_shorten_start_dim }
3988     \dim_set:Nn \l_@@_xdots_shorten_end_dim
3989         { 0.6 \l_@@_xdots_shorten_end_dim }
3990 }

```

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`.

```

3991 \NewDocumentCommand \@@_CodeAfter_keys: { O { } }
3992     { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

```

```

3993 \cs_new_protected:Npn \@@_create_alias_nodes:
3994 {
3995     \int_step_inline:nn { \c@iRow }
3996     {
3997         \pgfnodealias
3998             { \l_@@_name_str - ##1 - last }
3999             { \@@_env: - ##1 - \int_use:N \c@jCol }
4000     }
4001     \int_step_inline:nn { \c@jCol }
4002     {
4003         \pgfnodealias
4004             { \l_@@_name_str - last - ##1 }
4005             { \@@_env: - \int_use:N \c@iRow - ##1 }
4006     }
4007     \pgfnodealias
4008         { \l_@@_name_str - last - last }
4009         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
4010 }

```

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).

```

4011 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4012 {
4013     \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4014         { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4015 }

```

The following command must *not* be protected.

```

4016 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4017 {
4018     { #1 }
4019     { #2 }
4020     {
4021         \int_compare:nNnTF { #3 } > { 98 }
4022             { \int_use:N \c@iRow }
4023             { #3 }
4024 }

```

```

4025 {
4026   \int_compare:nNnTF { #4 } > { 98 }
4027     { \int_use:N \c@jCol }
4028     { #4 }
4029   }
4030   { #5 }
4031 }

```

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).

```

4032 \hook_gput_code:nnn { begindocument } { . }
4033 {
4034   \cs_new_protected:Npe \@@_draw_dotted_lines:
4035   {
4036     \c_@@_pgfortikzpicture_tl
4037     \@@_draw_dotted_lines_i:
4038     \c_@@_endpgfortikzpicture_tl
4039   }
4040 }

```

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

```

4041 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4042 {
4043   \pgfrememberpicturepositiononpagetrue
4044   \pgf@relevantforpicturesizefalse
4045   \g_@@_HVdotsfor_lines_tl
4046   \g_@@_Vdots_lines_tl
4047   \g_@@_Ddots_lines_tl
4048   \g_@@_Iddots_lines_tl
4049   \g_@@_Cdots_lines_tl
4050   \g_@@_Ldots_lines_tl
4051 }

4052 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4053 {
4054   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4055   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4056 }

```

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

```

4057 \pgfdeclareshape { @@_diag_node }
4058 {
4059   \savedanchor { \five }
4060   {
4061     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4062     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4063   }
4064   \anchor { 5 } { \five }
4065   \anchor { center } { \pgfpointorigin }
4066   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4067   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4068   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4069   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4070   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4071   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4072   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4073   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4074   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4075   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4076 }

```

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).

```

4077 \cs_new_protected:Npn \@@_create_diag_nodes:
4078 {
4079     \pgfpicture
4080     \pgfrememberpicturepositiononpagetrue
4081     \int_step_inline:nn { \int_max:nn { \c@iRow } { \c@jCol } }
4082     {
4083         \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4084         \dim_set_eq:NN \l_tmpa_dim \pgf@x
4085         \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4086         \dim_set_eq:NN \l_tmpb_dim \pgf@y
4087         \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4088         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4089         \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4090         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4091         \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.

```

4092 \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4093 \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4094 \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4095 \str_if_empty:NF \l_@@_name_str
4096     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4097 }

```

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

```

4098 \int_set:Nn \l_tmpa_int { \int_max:nn { \c@iRow } { \c@jCol } + 1 }
4099 \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4100 \dim_set_eq:NN \l_tmpa_dim \pgf@y
4101 \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4102 \pgfcordinate
4103     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4104 \pgfnodealias
4105     { \@@_env: - last }
4106     { \@@_env: - \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
4107 \str_if_empty:NF \l_@@_name_str
4108     {
4109         \pgfnodealias
4110             { \l_@@_name_str - \int_use:N \l_tmpa_int }
4111             { \@@_env: - \int_use:N \l_tmpa_int }
4112         \pgfnodealias
4113             { \l_@@_name_str - last }
4114             { \@@_env: - last }
4115     }
4116 \endpgfpicture
4117 }

```

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.

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

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

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

This command computes:

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

```
4118 \cs_new_protected:Npn \@_find_extremities_of_line:nnnn #1 #2 #3 #4
4119 {
```

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

```
4120 \cs_set_nopar:cpn { @_ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4121 \int_set:Nn \l @_initial_i_int { #1 }
4122 \int_set:Nn \l @_initial_j_int { #2 }
4123 \int_set:Nn \l @_final_i_int { #1 }
4124 \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.

```
4125 \bool_set_false:N \l @_stop_loop_bool
4126 \bool_do_until:Nn \l @_stop_loop_bool
4127 {
4128     \int_add:Nn \l @_final_i_int { #3 }
4129     \int_add:Nn \l @_final_j_int { #4 }
4130     \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.

```
4131 \if_int_compare:w \l @_final_i_int > \l @_row_max_int
4132     \if_int_compare:w #3 = \c_one_int
4133         \bool_set_true:N \l @_final_open_bool
4134     \else:
4135         \if_int_compare:w \l @_final_j_int > \l @_col_max_int
4136             \bool_set_true:N \l @_final_open_bool
4137         \fi:
4138     \fi:
4139 \else:
4140     \if_int_compare:w \l @_final_j_int < \l @_col_min_int
4141         \if_int_compare:w #4 = -1
4142             \bool_set_true:N \l @_final_open_bool
4143         \fi:
4144     \else:
4145         \if_int_compare:w \l @_final_j_int > \l @_col_max_int
4146             \if_int_compare:w #4 = \c_one_int
4147                 \bool_set_true:N \l @_final_open_bool
4148             \fi:
4149         \fi:
4150     \fi:
```

```
4152     \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.

```
4153     {
```

We do a step backwards.

```
4154         \int_sub:Nn \l_@@_final_i_int { #3 }
4155         \int_sub:Nn \l_@@_final_j_int { #4 }
4156         \bool_set_true:N \l_@@_stop_loop_bool
4157     }
```

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`.

```
4158     {
4159         \cs_if_exist:cTF
4160         {
4161             @@ _ dotted _
4162             \int_use:N \l_@@_final_i_int -
4163             \int_use:N \l_@@_final_j_int
4164         }
4165         {
4166             \int_sub:Nn \l_@@_final_i_int { #3 }
4167             \int_sub:Nn \l_@@_final_j_int { #4 }
4168             \bool_set_true:N \l_@@_final_open_bool
4169             \bool_set_true:N \l_@@_stop_loop_bool
4170         }
4171     }
4172     {
4173         \cs_if_exist:cTF
4174         {
4175             pgf @ sh @ ns @ \@@_env:
4176             - \int_use:N \l_@@_final_i_int
4177             - \int_use:N \l_@@_final_j_int
4178         }
4179         { \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.

```
4179     {
4180         \cs_set_nopar:cpn
4181         {
4182             @@ _ dotted _
4183             \int_use:N \l_@@_final_i_int -
4184             \int_use:N \l_@@_final_j_int
4185         }
4186         { }
4187     }
4188 }
4189 }
```

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

```
4191     \bool_set_false:N \l_@@_stop_loop_bool
```

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

```
4192     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4193     \bool_do_until:Nn \l_@@_stop_loop_bool
4194     {
4195         \int_sub:Nn \l_@@_initial_i_int { #3 }
4196         \int_sub:Nn \l_@@_initial_j_int { #4 }
4197         \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.

```

4198     \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4199         \if_int_compare:w #3 = \c_one_int
4200             \bool_set_true:N \l_@@_initial_open_bool
4201         \else:
4202
4203 \l_tmpa_int contains \l_@@_col_min_int - 1 (only for efficiency).
4204         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4205             \bool_set_true:N \l_@@_initial_open_bool
4206             \fi:
4207         \fi:
4208     \else:
4209         \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4210             \if_int_compare:w #4 = \c_one_int
4211                 \bool_set_true:N \l_@@_initial_open_bool
4212             \fi:
4213         \else:
4214             \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4215                 \if_int_compare:w #4 = -1
4216                     \bool_set_true:N \l_@@_initial_open_bool
4217                 \fi:
4218             \fi:
4219             \fi:
4220             \fi:
4221             \bool_if:NTF \l_@@_initial_open_bool
4222             {
4223                 \int_add:Nn \l_@@_initial_i_int { #3 }
4224                 \int_add:Nn \l_@@_initial_j_int { #4 }
4225                 \bool_set_true:N \l_@@_stop_loop_bool
4226             }
4227             {
4228                 \cs_if_exist:cTF
4229                 {
4230                     @@ _ dotted _
4231                     \int_use:N \l_@@_initial_i_int -
4232                     \int_use:N \l_@@_initial_j_int
4233                 }
4234                 {
4235                     \int_add:Nn \l_@@_initial_i_int { #3 }
4236                     \int_add:Nn \l_@@_initial_j_int { #4 }
4237                     \bool_set_true:N \l_@@_initial_open_bool
4238                     \bool_set_true:N \l_@@_stop_loop_bool
4239                 }
4240                 {
4241                     \cs_if_exist:cTF
4242                     {
4243                         pgf @ sh @ ns @ \@@_env:
4244                         - \int_use:N \l_@@_initial_i_int
4245                         - \int_use:N \l_@@_initial_j_int
4246                     }
4247                     {
4248                         \bool_set_true:N \l_@@_stop_loop_bool
4249                         {
4250                             \cs_set_nopar:cpn
4251                             {
4252                                 @@ _ dotted _
4253                                 \int_use:N \l_@@_initial_i_int -
4254                                 \int_use:N \l_@@_initial_j_int
4255                             }
4256                         }
4257                     }
4258                 }
4259             }
4260         }
4261     }
4262 }
```

```
4257 }
```

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.

```
4258 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4259 {
4260     { \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`.

```
4261     { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4262     { \int_use:N \l_@@_final_i_int }
4263     { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4264     { }
4265 }
4266 }
```

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.

```
4267 \cs_new_protected:Npn \@@_open_shorten:
4268 {
4269     \bool_if:NT \l_@@_initial_open_bool
4270         { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4271     \bool_if:NT \l_@@_final_open_bool
4272         { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4273 }
```

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).

```
4274 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4275 {
4276     \int_set_eq:NN \l_@@_row_min_int \c_one_int
4277     \int_set_eq:NN \l_@@_col_min_int \c_one_int
4278     \int_set_eq:NN \l_@@_row_max_int \c@iRow
4279     \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`.

```
4280 \seq_if_empty:NF \g_@@_submatrix_seq
4281 {
4282     \seq_map_inline:Nn \g_@@_submatrix_seq
4283         { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4284     }
4285 }
```

#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.

```

4286 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4287 {
4288     \if_int_compare:w #3 > #1
4289     \else:
4290         \if_int_compare:w #1 > #5
4291     \else:
4292         \if_int_compare:w #4 > #2
4293     \else:
4294         \if_int_compare:w #2 > #6
4295     \else:
4296         \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4297         \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4298         \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4299         \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4300     \fi:
4301     \fi:
4302     \fi:
4303     \fi:
4304 }

4305 \cs_new_protected:Npn \@@_set_initial_coords:
4306 {
4307     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4308     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4309 }
4310 \cs_new_protected:Npn \@@_set_final_coords:
4311 {
4312     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4313     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4314 }
4315 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4316 {
4317     \pgfpointanchor
4318     {
4319         \@@_env:
4320         - \int_use:N \l_@@_initial_i_int
4321         - \int_use:N \l_@@_initial_j_int
4322     }
4323     { #1 }
4324     \@@_set_initial_coords:
4325 }
4326 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4327 {
4328     \pgfpointanchor
4329     {
4330         \@@_env:
4331         - \int_use:N \l_@@_final_i_int
4332         - \int_use:N \l_@@_final_j_int
4333     }
4334     { #1 }
4335     \@@_set_final_coords:
4336 }

```

```

4337 \cs_new_protected:Npn \@@_open_x_initial_dim:
4338 {
4339     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4340     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4341     {
4342         \cs_if_exist:cT
4343             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4344             {
4345                 \pgfpointanchor
4346                     { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4347                     { west }
4348                 \dim_set:Nn \l_@@_x_initial_dim
4349                     { \dim_min:nn { \l_@@_x_initial_dim } { \pgf@x } }
4350             }
4351     }

```

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

```

4352 \dim_compare:nNnT { \l_@@_x_initial_dim } = { \c_max_dim }
4353 {
4354     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4355     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4356     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4357 }
4358 }

4359 \cs_new_protected:Npn \@@_open_x_final_dim:
4360 {
4361     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4362     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4363     {
4364         \cs_if_exist:cT
4365             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4366             {
4367                 \pgfpointanchor
4368                     { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4369                     { east }
4370                 \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
4371                     { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4372             }
4373     }

```

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

```

4374 \dim_compare:nNnT { \l_@@_x_final_dim } = { - \c_max_dim }
4375 {
4376     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4377     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4378     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4379 }
4380

```

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.

```

4381 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4382 {
4383     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4384     \cs_if_free:cT { @_ dotted _ #1 - #2 }
4385     {
4386         \@@_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.

```

4387 \group_begin:
4388     \@@_open_shorten:

```

```

4389     \int_if_zero:nTF { #1 }
4390         { \color { nicematrix-first-row } }
4391         {

```

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.

```

4392     \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4393         { \color { nicematrix-last-row } }
4394         }
4395         \keys_set:nn { nicematrix / xdots } { #3 }
4396         \color:o \l_@@_xdots_color_tl
4397         \actually_draw_Ldots:
4398         \group_end:
4399     }
4400 }

```

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`.

```

4401 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4402     {
4403         \bool_if:NTF \l_@@_initial_open_bool
4404             {
4405                 \@@_open_x_initial_dim:
4406                 \qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4407                 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4408             }
4409             \@@_set_initial_coords_from_anchor:n { base-east }
4410         \bool_if:NTF \l_@@_final_open_bool
4411             {
4412                 \@@_open_x_final_dim:
4413                 \qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4414                 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4415             }
4416             \@@_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.

```

4417 \bool_lazy_all:nTF
4418     {
4419         \l_@@_initial_open_bool
4420         \l_@@_final_open_bool
4421         { \int_compare_p:nNn { \l_@@_initial_i_int } = { \l_@@_last_row_int } }
4422     }
4423     {
4424         \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4425         \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4426     }

```

We raise the line of a quantity equal to the radius of the dots because we want the dots really “on” the line of text. Of course, maybe we should not do that when the option `line-style` is used (?).

```

4427     {
4428         \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim

```

```

4429     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4430   }
4431 \@@_draw_line:
4432 }
```

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.

```

4433 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4434 {
4435   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4436   \cs_if_free:cT { @_ dotted _ #1 - #2 }
4437   {
4438     \@@_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.

```

4439 \group_begin:
4440   \@@_open_shorten:
4441   \int_if_zero:nTF { #1 }
4442     { \color { nicematrix-first-row } }
4443 }
```

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.

```

4444   \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4445     { \color { nicematrix-last-row } }
4446   }
4447   \keys_set:nn { nicematrix / xdots } { #3 }
4448   \@@_color:o \l_@@_xdots_color_tl
4449   \@@_actually_draw_Cdots:
4450   \group_end:
4451 }
4452 }
```

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`.

```

4453 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4454 {
4455   \bool_if:NTF \l_@@_initial_open_bool
4456     { \@@_open_x_initial_dim: }
4457     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4458   \bool_if:NTF \l_@@_final_open_bool
4459     { \@@_open_x_final_dim: }
4460     { \@@_set_final_coords_from_anchor:n { mid-west } }
4461   \bool_lazy_and:nnTF
4462     { \l_@@_initial_open_bool }
4463     { \l_@@_final_open_bool }
4464   {
4465     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4466     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4467     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4468     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
```

```

4469 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4470 }
4471 {
4472     \bool_if:NT \l_@@_initial_open_bool
4473         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4474     \bool_if:NT \l_@@_final_open_bool
4475         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4476 }
4477 \@@_draw_line:
4478 }

4479 \cs_new_protected:Npn \@@_open_y_initial_dim:
4480 {
4481     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4482     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4483     {
4484         \cs_if_exist:cT
4485             { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4486         {
4487             \pgfpointanchor
4488                 { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4489                 { north }
4490             \dim_compare:nNnT { \pgf@y } > { \l_@@_y_initial_dim }
4491                 { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4492         }
4493     }
4494     \dim_compare:nNnT { \l_@@_y_initial_dim } = { - \c_max_dim }
4495     {
4496         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4497         \dim_set:Nn \l_@@_y_initial_dim
4498         {
4499             \fp_to_dim:n
4500             {
4501                 \pgf@y
4502                 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4503             }
4504         }
4505     }
4506 }

4507 \cs_new_protected:Npn \@@_open_y_final_dim:
4508 {
4509     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4510     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4511     {
4512         \cs_if_exist:cT
4513             { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4514         {
4515             \pgfpointanchor
4516                 { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4517                 { south }
4518             \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
4519                 { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4520         }
4521     }
4522     \dim_compare:nNnT { \l_@@_y_final_dim } = { \c_max_dim }
4523     {
4524         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4525         \dim_set:Nn \l_@@_y_final_dim
4526             { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4527     }
4528 }

```

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.

```

4529 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4530 {
4531     \@@_adjust_to_submatrix:n { #1 } { #2 }
4532     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4533     {
4534         \@@_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.

```

4535 \group_begin:
4536     \@@_open_shorten:
4537     \int_if_zero:nTF { #2 }
4538     {
4539         \color { nicematrix-first-col } }
4540         \int_compare:nNnT { #2 } = { \l_@@_last_col_int }
4541             { \color { nicematrix-last-col } }
4542     }
4543     \keys_set:nn { nicematrix / xdots } { #3 }
4544     \color:o \l_@@_xdots_color_tl
4545     \bool_if:NTF \l_@@_Vbrace_bool
4546         { \@@_actually_draw_Vbrace: }
4547         { \@@_actually_draw_Vdots: }
4548     \group_end:
4549 }
4550

```

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`.

```

4551 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4552 {
4553     \bool_lazy_and:nnTF { \l_@@_initial_open_bool } { \l_@@_final_open_bool }
4554         { \@@_actually_draw_Vdots_i: }
4555         { \@@_actually_draw_Vdots_ii: }
4556     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4557     \@@_draw_line:
4558 }

```

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

```

4559 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4560 {
4561     \@@_open_y_initial_dim:
4562     \@@_open_y_final_dim:
4563     \int_if_zero:nTF { \l_@@_initial_j_int }

```

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

```

4564 {
4565     \@@_qpoint:n { col - 1 }
4566     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4567     \dim_sub:Nn \l_@@_x_initial_dim
4568         { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4569 }
4570 {

```

```

4571 \bool_lazy_and:nNF
4572   { \int_compare_p:nNn { \l_@@_last_col_int } > { -2 } }
4573   {
4574     \int_compare_p:nNn
4575       { \l_@@_initial_j_int } = { \g_@@_col_total_int }
4576   }

```

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

```

4577   {
4578     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4579     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4580     \dim_add:Nn \l_@@_x_initial_dim
4581       { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4582   }

```

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

```

4583   {
4584     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4585     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4586     \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4587     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4588   }
4589 }
4590 }

```

The command `\@@_draw_line:` is in `\@@_actually_draw_Vdots:`

Now, the dotted line is *not* open on both sides (maybe open on only one side).

The main task is to determine the x -value of the dotted line to draw.

The boolean `\l_tmpa_bool` will indicate whether the column is of type 1 or may be considered as if.

```

4591 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4592   {
4593     \bool_set_false:N \l_tmpa_bool
4594     \bool_if:NTF \l_@@_initial_open_bool
4595     {
4596       \bool_if:NTF \l_@@_final_open_bool
4597       {
4598         \@@_set_initial_coords_from_anchor:n { south-west }
4599         \@@_set_final_coords_from_anchor:n { north-west }
4600         \bool_set:Nn \l_tmpa_bool
4601           {
4602             \dim_compare_p:nNn
4603               { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4604           }
4605       }
4606     }
4607   }

```

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

```

4607 \bool_if:NTF \l_@@_initial_open_bool
4608   {
4609     \@@_open_y_initial_dim:
4610     \@@_set_final_coords_from_anchor:n { north }
4611     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4612   }
4613   {
4614     \@@_set_initial_coords_from_anchor:n { south }
4615     \bool_if:NTF \l_@@_final_open_bool
4616       { \@@_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.

```

4617   {
4618     \@@_set_final_coords_from_anchor:n { north }
4619     \dim_compare:nNnf { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4620       {

```

```

4621     \dim_set:Nn \l_@@_x_initial_dim
4622     {
4623         \bool_if:NTF \l_tmpa_bool { \dim_min:nn } { \dim_max:nn }
4624             \l_@@_x_initial_dim \l_@@_x_final_dim
4625     }
4626 }
4627 }
4628 }
4629 }
```

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.`

```

4630 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4631 {
4632     \bool_if:NTF \l_@@_initial_open_bool
4633         { \@@_open_y_initial_dim: }
4634         { \@@_set_initial_coords_from_anchor:n { south } }
4635     \bool_if:NTF \l_@@_final_open_bool
4636         { \@@_open_y_final_dim: }
4637         { \@@_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.

```

4638 \int_if_zero:nTF { \l_@@_initial_j_int }
4639 {
4640     \@@_qpoint:n { col - 1 }
4641     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4642     \dim_sub:Nn \l_@@_x_initial_dim
4643         { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4644 }
```

Elsewhere, the brace must be drawn left flush.

```

4645 {
4646     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4647     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4648     \dim_add:Nn \l_@@_x_initial_dim
4649         { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4650 }
```

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

```

4651 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4652 \@@_draw_line:
4653 }
4654 \cs_new:Npn \@@_colsep:
4655     { \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.

```
4656 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4657 {
4658     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4659     \cs_if_free:cT { @_ dotted _ #1 - #2 }
4660     {
4661         \@@_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.

```
4662 \group_begin:
4663     \@@_open_shorten:
4664     \keys_set:nn { nicematrix / xdots } { #3 }
4665     \@@_color:o \l_@_xdots_color_tl
4666     \@@_actually_draw_Ddots:
4667     \group_end:
4668 }
4669 }
```

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.`

```
4670 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4671 {
4672     \bool_if:NTF \l_@_initial_open_bool
4673     {
4674         \@@_open_y_initial_dim:
4675         \@@_open_x_initial_dim:
4676     }
4677     { \@@_set_initial_coords_from_anchor:n { south-east } }
4678     \bool_if:NTF \l_@_final_open_bool
4679     {
4680         \@@_open_x_final_dim:
4681         \dim_set_eq:NN \l_@_x_final_dim \pgf@x
4682     }
4683     { \@@_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.

```
4684     \bool_if:NT \l_@_parallelize_diags_bool
4685     {
4686         \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).

```
4687     \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.

```
4688     {
```

```

4689         \dim_gset:Nn \g_@@_delta_x_one_dim
4690             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4691         \dim_gset:Nn \g_@@_delta_y_one_dim
4692             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4693     }

```

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`.

```

4694     {
4695         \dim_compare:nNnF { \g_@@_delta_x_one_dim } = { \c_zero_dim }
4696             {
4697                 \dim_set:Nn \l_@@_y_final_dim
4698                     {
4699                         \l_@@_y_initial_dim +
4700                         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4701                         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4702                     }
4703             }
4704         }
4705     }
4706     \@@_draw_line:
4707 }

```

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.

```

4708 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4709 {
4710     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4711     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4712     {
4713         \@@_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.

```

4714     \group_begin:
4715         \@@_open_shorten:
4716         \keys_set:nn { nicematrix / xdots } { #3 }
4717         \@@_color:o \l_@@_xdots_color_tl
4718         \@@_actually_draw_Iddots:
4719     \group_end:
4720 }
4721 }

```

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`.

```

4722 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4723 {
4724     \bool_if:NTF \l_@@_initial_open_bool
4725     {
4726         \@@_open_y_initial_dim:
4727         \@@_open_x_initial_dim:
4728     }

```

```

4729 { \@@_set_initial_coords_from_anchor:n { south-west } }
4730 \bool_if:NTF \l_@@_final_open_bool
4731 {
4732     \@@_open_y_final_dim:
4733     \@@_open_x_final_dim:
4734 }
4735 { \@@_set_final_coords_from_anchor:n { north-east } }
4736 \bool_if:NT \l_@@_parallelize_diags_bool
4737 {
4738     \int_gincr:N \g_@@_iddots_int
4739     \int_compare:nNnTF { \g_@@_iddots_int } = { \c_one_int }
4740     {
4741         \dim_gset:Nn \g_@@_delta_x_two_dim
4742             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4743         \dim_gset:Nn \g_@@_delta_y_two_dim
4744             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4745     }
4746     {
4747         \dim_compare:nNnF { \g_@@_delta_x_two_dim } = { \c_zero_dim }
4748         {
4749             \dim_set:Nn \l_@@_y_final_dim
4750             {
4751                 \l_@@_y_initial_dim +
4752                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4753                     \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4754             }
4755         }
4756     }
4757 }
4758 \@@_draw_line:
4759 }
```

17 The actual instructions for drawing the dotted lines with Tikz

The command `\@@_draw_line:` should be used in a `{pgfpicture}`. It has six implicit arguments:

- `\l_@@_x_initial_dim`
- `\l_@@_y_initial_dim`
- `\l_@@_x_final_dim`
- `\l_@@_y_final_dim`
- `\l_@@_initial_open_bool`
- `\l_@@_final_open_bool`

```

4760 \cs_new_protected:Npn \@@_draw_line:
4761 {
4762     \pgfrememberpicturepositiononpagetrue
4763     \pgf@relevantforpicturesizefalse
4764     \bool_lazy_or:nnTF
4765         { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4766         { \l_@@_dotted_bool }
4767         { \@@_draw_standard_dotted_line: }
4768         { \@@_draw_unstandard_dotted_line: }
4769 }
```

We have to do a special construction with `\exp_args:N` to be able to put in the list of options in the correct place in the Tikz instruction.

```

4770 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4771 {
4772     \begin { scope }
4773         \@@_draw_unstandard_dotted_line:o
4774             { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4775 }

```

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 diretly `\l_@@_xdots_color_tl`).

The argument of `\@@_draw_unstandard_dotted_line:n` is, in fact, the list of options.

```

4776 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4777 {
4778     \@@_draw_unstandard_dotted_line:nooo
4779         { #1 }
4780         \l_@@_xdots_up_tl
4781         \l_@@_xdots_down_tl
4782         \l_@@_xdots_middle_tl
4783 }
4784 \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.

```

4785 \hook_gput_code:nnn { begindocument } { . }
4786 {
4787     \IfPackageLoadedT { tikz }
4788     {
4789         \tikzset
4790         {
4791             @@_node_above / .style = { sloped , above } ,
4792             @@_node_below / .style = { sloped , below } ,
4793             @@_node_middle / .style =
4794             {
4795                 sloped ,
4796                 inner_sep = \c_@@_innersep_middle_dim
4797             }
4798         }
4799     }
4800 }

4801 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4802 {

```

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.

```

4803 \dim_zero_new:N \l_@@_l_dim
4804 \dim_set:Nn \l_@@_l_dim
4805 {
4806     \fp_to_dim:n
4807     {
4808         sqrt
4809         (
4810             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4811             +
4812             ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4813         )
4814     }
4815 }

```

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.

```
4816 \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4817 {
4818     \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4819         \@@_draw_unstandard_dotted_line_i:
4820 }
```

If the key xdots/horizontal-labels has been used.

```
4821 \bool_if:NT \l_@@_xdots_h_labels_bool
4822 {
4823     \tikzset
4824     {
4825         @@_node_above / .style = { auto = left } ,
4826         @@_node_below / .style = { auto = right } ,
4827         @@_node_middle / .style = { inner sep = \c_@@_innersep_middle_dim }
4828     }
4829 }
4830 \tl_if_empty:nF { #4 }
4831     { \tikzset { @@_node_middle / .append style = { fill = white } } }
4832 \dim_zero:N \l_tmpa_dim
4833 \dim_zero:N \l_tmpb_dim
4834 \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
4835 {
```

We test whether the brace is vertical or horizontal.

```
4836 \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4837     { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
4838     { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
4839 }
4840 {
4841     \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
4842     {
4843         \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4844             { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
4845             { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
4846     }
4847 }
4848 \use:e
4849 {
4850     \exp_not:N \begin { scope }
4851         [ shift = { (\dim_use:N \l_tmpa_dim, \dim_use:N \l_tmpb_dim) } ]
4852     }
4853 \draw
4854 [ #1 ]
4855     ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
4856     -- node [ @@_node_middle] { $ \scriptstyle #4 $ }
4857     node [ @@_node_below ] { $ \scriptstyle #3 $ }
4858     node [ @@_node_above ] { $ \scriptstyle #2 $ }
4859     ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4860 \end { scope }
4861 \end { scope }
4862 }
4863 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4864 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4865 {
4866     \dim_set:Nn \l_tmpa_dim
4867     {
4868         \l_@@_x_initial_dim
4869         + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4870         * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
```

```

4871   }
4872   \dim_set:Nn \l_tmpb_dim
4873   {
4874     \l_@@_y_initial_dim
4875     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4876     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4877   }
4878   \dim_set:Nn \l_@@_tmpc_dim
4879   {
4880     \l_@@_x_final_dim
4881     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4882     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4883   }
4884   \dim_set:Nn \l_@@_tmpd_dim
4885   {
4886     \l_@@_y_final_dim
4887     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4888     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4889   }
4890   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4891   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4892   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4893   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4894 }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```

4895 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4896 {
4897   \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.

```

4898   \dim_zero_new:N \l_@@_l_dim
4899   \dim_set:Nn \l_@@_l_dim
5000   {
5001     \fp_to_dim:n
5002     {
5003       sqrt
5004       (
5005         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5006         +
5007         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5008       )
5009     }
5010   }

```

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.

```

4911   \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4912   {
4913     \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4914     { \@@_draw_standard_dotted_line_i: }
4915   }
4916   \group_end:
4917   \bool_lazy_all:nF
4918   {
4919     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4920     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4921     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4922   }

```

```

4923     { \c_@@_labels_standard_dotted_line: }
4924   }
4925 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4926 \cs_new_protected:Npn \c_@@_draw_standard_dotted_line_i:
4927   {

```

The number of dots will be $\l_l_{tmpa_int} + 1$.

```

4928   \int_set:Nn \l_l_tmpa_int
4929   {
4930     \dim_ratio:nn
4931     {
4932       \l_l_@@_l_dim
4933       - \l_l_@@_xdots_shorten_start_dim
4934       - \l_l_@@_xdots_shorten_end_dim
4935     }
4936     { \l_l_@@_xdots_inter_dim }
4937   }

```

The dimensions $\l_l_{tmpa_dim}$ and $\l_l_{tmpb_dim}$ are the coordinates of the vector between two dots in the dotted line.

```

4938   \dim_set:Nn \l_l_tmpa_dim
4939   {
4940     ( \l_l_@@_x_final_dim - \l_l_@@_x_initial_dim ) *
4941     \dim_ratio:nn \l_l_@@_xdots_inter_dim \l_l_@@_l_dim
4942   }
4943   \dim_set:Nn \l_l_tmpb_dim
4944   {
4945     ( \l_l_@@_y_final_dim - \l_l_@@_y_initial_dim ) *
4946     \dim_ratio:nn \l_l_@@_xdots_inter_dim \l_l_@@_l_dim
4947   }

```

In the loop over the dots, the dimensions $\l_l_@@_x_initial_dim$ and $\l_l_@@_y_initial_dim$ will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

4948   \dim_gadd:Nn \l_l_@@_x_initial_dim
4949   {
4950     ( \l_l_@@_x_final_dim - \l_l_@@_x_initial_dim ) *
4951     \dim_ratio:nn
4952     {
4953       \l_l_@@_l_dim - \l_l_@@_xdots_inter_dim * \l_l_tmpa_int
4954       + \l_l_@@_xdots_shorten_start_dim - \l_l_@@_xdots_shorten_end_dim
4955     }
4956     { 2 \l_l_@@_l_dim }
4957   }
4958   \dim_gadd:Nn \l_l_@@_y_initial_dim
4959   {
4960     ( \l_l_@@_y_final_dim - \l_l_@@_y_initial_dim ) *
4961     \dim_ratio:nn
4962     {
4963       \l_l_@@_l_dim - \l_l_@@_xdots_inter_dim * \l_l_tmpa_int
4964       + \l_l_@@_xdots_shorten_start_dim - \l_l_@@_xdots_shorten_end_dim
4965     }
4966     { 2 \l_l_@@_l_dim }
4967   }
4968 \pgf@relevantforpicturesizefalse
4969 \int_step_inline:nnn { \c_zero_int } { \l_l_tmpa_int }
4970   {
4971     \pgfpathcircle
4972     { \pgfpoint \l_l_@@_x_initial_dim \l_l_@@_y_initial_dim }
4973     { \l_l_@@_xdots_radius_dim }
4974     \dim_add:Nn \l_l_@@_x_initial_dim \l_l_tmpa_dim
4975     \dim_add:Nn \l_l_@@_y_initial_dim \l_l_tmpb_dim
4976   }
4977   \pgfusepathqfill
4978 }

```

```

4979 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
4980 {
4981     \pgfscope
4982     \pgftransformshift
4983     {
4984         \pgfpointlineattime { 0.5 }
4985         { \pgfpoint {\l_@@_x_initial_dim} {\l_@@_y_initial_dim} }
4986         { \pgfpoint {\l_@@_x_final_dim} {\l_@@_y_final_dim} }
4987     }
4988     \fp_set:Nn \l_tmpa_fp
4989     {
4990         \atand
4991         (
4992             \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4993             \l_@@_x_final_dim - \l_@@_x_initial_dim
4994         )
4995     }
4996     \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4997     \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4998     \tl_if_empty:NF \l_@@_xdots_middle_tl
4999     {
5000         \begin { pgfscope }
5001             \pgfset { inner~sep = \c_@@_innersep_middle_dim }
5002             \pgfnode
5003                 { rectangle }
5004                 { center }
5005                 {
5006                     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5007                     {
5008                         $ \% $
5009                         \scriptstyle \l_@@_xdots_middle_tl
5010                         $ \% $
5011                     }
5012                 }
5013                 {
5014                     \pgfsetfillcolor { white }
5015                     \pgfusepath { fill }
5016                 }
5017             \end { pgfscope }
5018         }
5019     \tl_if_empty:NF \l_@@_xdots_up_tl
5020     {
5021         \pgfnode
5022             { rectangle }
5023             { south }
5024             {
5025                 \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5026                 {
5027                     $ \% $
5028                     \scriptstyle \l_@@_xdots_up_tl
5029                     $ \% $
5030                 }
5031             }
5032             {
5033                 \pgfusepath { }
5034             }
5035         }
5036     \tl_if_empty:NF \l_@@_xdots_down_tl
5037     {
5038         \pgfnode
5039             { rectangle }
5040             { north }
5041             {

```

```

5042     \rotatebox { \fp_eval:n { - \l_tmpa_fp } } \\
5043     {
5044         $ \% $ \\
5045         \scriptstyle \l_@@_xdots_down_tl \\
5046         $ \% $ \\
5047     } \\
5048 } \\
5049 \{ \\
5050 \{ \pgfusepath \} \\
5051 \} \\
5052 \endpgfscope \\
5053 }

```

18 User commands available in the new environments

The commands `\@@_Ldots:`, `\@@_Cdots:`, `\@@_Vdots:`, `\@@_Ddots:` and `\@@_Idots:` will be linked to `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots` and `\Idots` 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.

```

5054 \hook_gput_code:nnn { begindocument } { . } \\
5055 {

```

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that's why we are in a `\AtBeginDocument`).

```

5056 \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } } \\
5057 \cs_new_protected:Npn \@@_Ldots: \\
5058   { \@@_collect_options:n { \@@_Ldots_i } } \\
5059 \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl \\
5060   { \\
5061     \int_if_zero:nTF { \c@jCol } \\
5062       { \@@_error:nn { in-first-col } { \Ldots } } \\
5063     { \\
5064       \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int } \\
5065         { \@@_error:nn { in-last-col } { \Ldots } } \\
5066       { \\
5067         \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots } \\
5068           { #1 , down = #2 , up = #3 , middle = #4 } \\
5069         } \\
5070       } \\
5071     \bool_if:NF \l_@@_nullify_dots_bool \\
5072       { \phantom { \ensuremath { \oldldots } } } \\
5073     \bool_gset_true:N \g_@@_empty_cell_bool \\
5074   } \\
5075 \\
5076 \cs_new_protected:Npn \@@_Cdots: \\
5077   { \@@_collect_options:n { \@@_Cdots_i } } \\
5078 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl \\
5079   { \\
5080     \int_if_zero:nTF { \c@jCol } \\
5081       { \@@_error:nn { in-first-col } { \Cdots } } \\
5082       { \\
5083         \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int } \\
5084           { \@@_error:nn { in-last-col } { \Cdots } } \\
5085         } \\
5086       } \\
5087     } \\
5088   }

```

```

5084         {
5085             \@@_instruction_of_type:nmn { \c_false_bool } { Cdots }
5086             { #1 , down = #2 , up = #3 , middle = #4 }
5087         }
5088     }
5089 \bool_if:NF \l_@@_nullify_dots_bool
5090     { \phantom { \ensuremath { \oldcdots } } } }
5091 \bool_gset_true:N \g_@@_empty_cell_bool
5092 }

5093 \cs_new_protected:Npn \@@_Vdots:
5094     { \@@_collect_options:n { \@@_Vdots_i } }
5095 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5096     {
5097         \int_if_zero:nTF { \c@iRow }
5098             { \@@_error:nn { in-first-row } { \Vdots } }
5099         {
5100             \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
5101                 { \@@_error:nn { in-last-row } { \Vdots } }
5102             {
5103                 \@@_instruction_of_type:nnn { \c_false_bool } { Vdots }
5104                     { #1 , down = #2 , up = #3 , middle = #4 }
5105             }
5106         }
5107     \bool_if:NF \l_@@_nullify_dots_bool
5108         { \phantom { \ensuremath { \oldvdots } } } }
5109     \bool_gset_true:N \g_@@_empty_cell_bool
5110 }

5111 \cs_new_protected:Npn \@@_Ddots:
5112     { \@@_collect_options:n { \@@_Ddots_i } }
5113 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5114     {
5115         \int_case:nnF \c@iRow
5116             {
5117                 0           { \@@_error:nn { in-first-row } { \Ddots } }
5118                 \l_@@_last_row_int { \@@_error:nn { in-last-row } { \Ddots } }
5119             }
5120         {
5121             \int_case:nnF \c@jCol
5122                 {
5123                     0           { \@@_error:nn { in-first-col } { \Ddots } }
5124                     \l_@@_last_col_int { \@@_error:nn { in-last-col } { \Ddots } }
5125                 }
5126             {
5127                 \keys_set_known:nn { nicematrix / Ddots } { #1 }
5128                 \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5129                     { #1 , down = #2 , up = #3 , middle = #4 }
5130             }
5131         }
5132     \bool_if:NF \l_@@_nullify_dots_bool
5133         { \phantom { \ensuremath { \oldddots } } } }
5134     \bool_gset_true:N \g_@@_empty_cell_bool
5135 }

5137 \cs_new_protected:Npn \@@_Iddots:
5138     { \@@_collect_options:n { \@@_Iddots_i } }
5139 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5140     {
5141         \int_case:nnF \c@iRow

```

```

5142 {
5143   0           { \@@_error:nn { in-first-row } { \Idots } }
5144   \l@@_last_row_int { \@@_error:nn { in-last-row } { \Idots } }
5145 }
5146 {
5147   \int_case:nnF \c@jCol
5148   {
5149     0           { \@@_error:nn { in-first-col } { \Idots } }
5150     \l@@_last_col_int { \@@_error:nn { in-last-col } { \Idots } }
5151   }
5152   {
5153     \keys_set_known:nn { nicematrix / Ddots } { #1 }
5154     \@@_instruction_of_type:nmm { \l@@_draw_first_bool } { Idots }
5155     { #1 , down = #2 , up = #3 , middle = #4 }
5156   }
5157 }
5158 \bool_if:NF \l@@_nullify_dots_bool
5159   { \phantom { \ensuremath { \oldidots } } }
5160   \bool_gset_true:N \g@@_empty_cell_bool
5161 }
5162 }
```

End of the `\AddToHook`.

Despite its name, the following set of keys will be used for `\Ddots` but also for `\Idots`.

```

5163 \keys_define:nn { nicematrix / Ddots }
5164 {
5165   draw-first .bool_set:N = \l@@_draw_first_bool ,
5166   draw-first .default:n = true ,
5167   draw-first .value_forbidden:n = true
5168 }
```

The command `\@@_Hspace:` will be linked to `\hspace` in `{NiceArray}`.

```

5169 \cs_new_protected:Npn \@@_Hspace:
5170 {
5171   \bool_gset_true:N \g@@_empty_cell_bool
5172   \hspace
5173 }
```

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`.

```
5174 \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`.

```

5175 \cs_new:Npn \@@_Hdotsfor:
5176 {
5177   \bool_lazy_and:nnTF
5178   { \int_if_zero_p:n { \c@jCol } }
5179   { \int_if_zero_p:n { \l@@_first_col_int } }
5180   {
5181     \bool_if:NTF \g@@_after_col_zero_bool
5182     {
5183       \multicolumn { 1 } { c } { }
5184       \@@_Hdotsfor_i:
5185     }
5186     { \@@_fatal:n { Hdotsfor-in-col~0 } }
5187   }
5188 }
```

```

5190     \@@_Hdotsfor_i:
5191   }
5192 }

```

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:`).

```

5193 \hook_gput_code:nnn { begindocument } { . }
5194 {

```

We don't put ! before the last optional argument for homogeneity with `\Cdots`, etc. which have only one optional argument.

```

5195 \cs_new_protected:Npn \@@_Hdotsfor_i:
5196   { \@@_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`).

```

5197 \tl_set_rescan:Nnn \l_tmpa_t1 { } { m m O { } E { _ ^ : } { { } { } { } } } }
5198 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_t1
5199 {
5200   \tl_gput_right:Ne \g_@@_HVdotsfor_lines_t1
5201   {
5202     \@@_Hdotsfor:nnnn
5203       { \int_use:N \c@iRow }
5204       { \int_use:N \c@jCol }
5205       { #2 }
5206       {
5207         #1 , #3 ,
5208         down = \exp_not:n { #4 } ,
5209         up = \exp_not:n { #5 } ,
5210         middle = \exp_not:n { #6 }
5211       }
5212     }
5213   \prg_replicate:nn { #2 - 1 }
5214   {
5215     &
5216     \multicolumn { 1 } { c } { }
5217     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5218   }
5219 }
5220 }

5221 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5222 {
5223   \bool_set_false:N \l_@@_initial_open_bool
5224   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5225 \int_set:Nn \l_@@_initial_i_int { #1 }
5226 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

5227 \int_compare:nNnTF { #2 } = { \c_one_int }
5228   {
5229     \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5230     \bool_set_true:N \l_@@_initial_open_bool
5231   }
5232   {
5233     \cs_if_exist:cTF
5234       {
5235         pgf @ sh @ ns @ \@@_env:
5236         - \int_use:N \l_@@_initial_i_int
5237         - \int_eval:n { #2 - 1 }
5238       }
5239     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }

```

```

5240      {
5241          \int_set:Nn \l_@@_initial_j_int { #2 }
5242          \bool_set_true:N \l_@@_initial_open_bool
5243      }
5244  }
5245  \int_compare:nNnTF { #2 + #3 - 1 } = { \c@jCol }
5246  {
5247      \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5248      \bool_set_true:N \l_@@_final_open_bool
5249  }
5250  {
5251      \cs_if_exist:cTF
5252      {
5253          pgf @ sh @ ns @ \@@_env:
5254          - \int_use:N \l_@@_final_i_int
5255          - \int_eval:n { #2 + #3 }
5256      }
5257      { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5258      {
5259          \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5260          \bool_set_true:N \l_@@_final_open_bool
5261      }
5262  }

5263 \group_begin:
5264 \@@_open_shorten:
5265 \int_if_zero:nTF { #1 }
5266     { \color { nicematrix-first-row } }
5267     {
5268         \int_compare:nNnT { #1 } = { \g_@@_row_total_int }
5269         { \color { nicematrix-last-row } }
5270     }
5271 \keys_set:nn { nicematrix / xdots } { #4 }
5272 \@@_color:o \l_@@_xdots_color_tl
5273 \@@_actually_draw_Ldots:
5274 \group_end:

```

We declare all the cells concerned by the `\Hdotsfor` as “dotted” (for the dotted lines created by `\Cdots`, `\Ldots`, etc., this job is done by `\@@_find_extremities_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5275 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5276     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5277 }

5278 \hook_gput_code:nnn { begindocument } { . }
5279 {
5280     \cs_new_protected:Npn \@@_Vdotsfor:
5281     { \@@_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`).

```

5282 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m O { } E { _ ^ : } { { } { } { } } }
5283 \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5284 {
5285     \bool_gset_true:N \g_@@_empty_cell_bool
5286     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5287     {
5288         \@@_Vdotsfor:nnnn
5289         { \int_use:N \c@iRow }
5290         { \int_use:N \c@jCol }
5291         { #2 }
5292         {
5293             #1 , #3 ,

```

```

5294     down = \exp_not:n { #4 } ,
5295     up = \exp_not:n { #5 } ,
5296     middle = \exp_not:n { #6 }
5297   }
5298 }
5299 }
5300 }
```

#1 is the number of row;
#2 is the number of column;
#3 is the numbers of rows which are involved;

```

5301 \cs_new_protected:Npn \Oo_Vdotsfor:nnnn #1 #2 #3 #4
5302 {
5303   \bool_set_false:N \l_@@_initial_open_bool
5304   \bool_set_false:N \l_@@_final_open_bool
```

For the column, it's easy.

```

5305   \int_set:Nn \l_@@_initial_j_int { #2 }
5306   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int
```

For the row, it's a bit more complicated.

```

5307   \int_compare:nNnTF { #1 } = { \c_one_int }
5308   {
5309     \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5310     \bool_set_true:N \l_@@_initial_open_bool
5311   }
5312   {
5313     \cs_if_exist:cTF
5314     {
5315       pgf @ sh @ ns @ \Oo_env:
5316       - \int_eval:n { #1 - 1 }
5317       - \int_use:N \l_@@_initial_j_int
5318     }
5319     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5320     {
5321       \int_set:Nn \l_@@_initial_i_int { #1 }
5322       \bool_set_true:N \l_@@_initial_open_bool
5323     }
5324   }
5325   \int_compare:nNnTF { #1 + #3 - 1 } = { \c@iRow }
5326   {
5327     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5328     \bool_set_true:N \l_@@_final_open_bool
5329   }
5330   {
5331     \cs_if_exist:cTF
5332     {
5333       pgf @ sh @ ns @ \Oo_env:
5334       - \int_eval:n { #1 + #3 }
5335       - \int_use:N \l_@@_final_j_int
5336     }
5337     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5338     {
5339       \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5340       \bool_set_true:N \l_@@_final_open_bool
5341     }
5342   }
5343   \group_begin:
5344   \Oo_open_shorten:
5345   \int_if_zero:nTF { #2 }
5346   { \color { nicematrix-first-col } }
5347   {
5348     \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
```

```

5349      { \color { nicematrix-last-col } }
5350    }
5351    \keys_set:nn { nicematrix / xdots } { #4 }
5352    \@@_color:o \l_@@_xdots_color_tl
5353    \bool_if:NTF \l_@@_Vbrace_bool
5354      { \@@_actually_draw_Vbrace: }
5355      { \@@_actually_draw_Vdots: }
5356  \group_end:

```

We declare all the cells concerned by the `\Vdots` 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).

```

5357  \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5358    { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5359  }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5360 \NewDocumentCommand \@@_rotate: { O { } }
5361 {
5362   \bool_gset_true:N \g_@@_rotate_bool
5363   \keys_set:nn { nicematrix / rotate } { #1 }
5364   \ignorespaces
5365 }

5366 \keys_define:nn { nicematrix / rotate }
5367 {
5368   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5369   c .value_forbidden:n = true ,
5370   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5371 }

```

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

```

5372 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5373 {
5374   \tl_if_empty:nTF { #2 }
5375     { #1 }
5376     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5377 }
5378 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5379   { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

¹⁴Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

With the following construction, the command `\@@_double_int_eval:n` is applied to both arguments before the application of `\@@_line_i:nn` (the construction uses the fact the `\@@_line_i:nn` is protected and that `\@@_double_int_eval:n` is fully expandable).

```
5380 \hook_gput_code:nnn { begindocument } { . }
5381 {
```

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that's why we are in a `\AtBeginDocument`).

```
5382 \tl_set_rescan:Nnn \l_tmpa_tl { }
5383 { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5384 \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5385 {
5386     \group_begin:
5387     \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5388     \@@_color:o \l_@@_xdots_color_tl
5389     \use:e
5390     {
5391         \@@_line_i:nn
5392             { \@@_double_int_eval:n #2 - \q_stop }
5393             { \@@_double_int_eval:n #3 - \q_stop }
5394     }
5395     \group_end:
5396 }
5397 }

5398 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5399 {
5400     \bool_set_false:N \l_@@_initial_open_bool
5401     \bool_set_false:N \l_@@_final_open_bool
5402     \bool_lazy_or:nnTF
5403         { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5404         { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5405         { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }
```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```
5406 { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5407 }

5408 \hook_gput_code:nnn { begindocument } { . }
5409 {
5410     \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5411     {
```

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:`.

```
5412     \c_@@_pgfortikzpicture_tl
5413     \@@_draw_line_ii:nn { #1 } { #2 }
5414     \c_@@_endpgfortikzpicture_tl
5415 }
5416 }
```

The following command *must* be protected (it's used in the construction of `\@@_draw_line_ii:nn`).

```
5417 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5418 {
5419     \pgfrememberpicturepositiononpagetrue
5420     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5421     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5422     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5423     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5424     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5425     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5426     \@@_draw_line:
5427 }
```

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_then: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_then: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.

```
5428 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNnT { \c@iRow } < }
5429 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNnF { \c@jCol } < }
```

`\@@_put_in_row_style` will be used several times in `\RowStyle`.

```
5430 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5431 {
5432     \tl_gput_right:Ne \g_@@_row_style_tl
5433 }
```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can't be replaced by a protected version of `\@@_if_row_less_than:nn`.

```
5434 \exp_not:N
5435 \@@_if_row_less_than:nn
5436 { \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`).

```
5437 {
5438     \exp_not:N
5439     \@@_if_col_greater_than:nn
5440     { \int_eval:n { \c@jCol } }
5441     { \exp_not:n { #1 } \scan_stop: }
5442 }
5443 }
5444 }
5445 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

5446 \keys_define:nn { nicematrix / RowStyle }
5447 {
5448     cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5449     cell-space-top-limit .value_required:n = true ,
5450     cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5451     cell-space-bottom-limit .value_required:n = true ,
5452     cell-space-limits .meta:n =
5453     {
5454         cell-space-top-limit = #1 ,
5455         cell-space-bottom-limit = #1 ,
5456     },
5457     color .tl_set:N = \l_@@_color_tl ,
5458     color .value_required:n = true ,
5459     bold .bool_set:N = \l_@@_bold_row_style_bool ,
5460     bold .default:n = true ,
```

```

5461 nb-rows .code:n =
5462   \str_if_eq:eeTF { #1 } { * }
5463   { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5464   { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5465 nb-rows .value_required:n = true ,
5466 fill .tl_set:N = \l_@@_fill_tl ,
5467 fill .value_required:n = true ,

```

In fine, the opacity will be applied by `\pgfsetfillopacity`.

```

5468 opacity .tl_set:N = \l_@@_opacity_tl ,
5469 opacity .value_required:n = true ,
5470 rowcolor .tl_set:N = \l_@@_fill_tl ,
5471 rowcolor .value_required:n = true ,
5472 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5473 rounded-corners .default:n = 4 pt ,
5474 unknown .code:n =
5475   \@@_unknown_key:nn
5476   { nicematrix / RowStyle }
5477   { Unknown~key~for~RowStyle }
5478 }

```

```

5479 \NewDocumentCommand \@@_RowStyle:n { O { } m }
5480 {
5481   \group_begin:
5482   \tl_clear:N \l_@@_fill_tl
5483   \tl_clear:N \l_@@_opacity_tl
5484   \tl_clear:N \l_@@_color_tl
5485   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5486   \dim_zero:N \l_@@_rounded_corners_dim
5487   \dim_zero:N \l_tmpa_dim
5488   \dim_zero:N \l_tmpb_dim
5489   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

If the key `fill` (or its alias `rowcolor`) has been used.

```

5490 \tl_if_empty:NF \l_@@_fill_tl
5491 {
5492   \@@_add_opacity_to_fill:
5493   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5494 }

```

The command `\@@_exp_color_arg:No` is *fully expandable*.

```

5495   \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5496   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5497   {
5498     \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5499     - *
5500   }
5501   { \dim_use:N \l_@@_rounded_corners_dim }
5502 }
5503 }
5504 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

5505 \dim_compare:nNnT { \l_tmpa_dim } > { \c_zero_dim }
5506 {
5507   \@@_put_in_row_style:e
5508   {
5509     \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5510   }

```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```

5511   \dim_set:Nn \l_@@_cell_space_top_limit_dim
5512   { \dim_use:N \l_tmpa_dim }
5513 }
5514 }
5515 }

```

\l_tmpb_dim is the value of the key `cell-space-bottom-limit` of \RowStyle.

```
5516   \dim_compare:nNnT { \l_tmpb_dim } > { \c_zero_dim }
5517   {
5518     \@@_put_in_row_style:e
5519     {
5520       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5521       {
5522         \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5523         { \dim_use:N \l_tmpb_dim }
5524       }
5525     }
5526   }
```

\l_@@_color_tl is the value of the key `color` of \RowStyle.

```
5527   \tl_if_empty:NF \l_@@_color_tl
5528   {
5529     \@@_put_in_row_style:e
5530     {
5531       \mode_leave_vertical:
5532       \color:n { \l_@@_color_tl }
5533     }
5534   }
```

\l_@@_bold_row_style_bool is the value of the key `bold`.

```
5535   \bool_if:NT \l_@@_bold_row_style_bool
5536   {
5537     \@@_put_in_row_style:n
5538     {
5539       \exp_not:n
5540       {
5541         \if_mode_math:
5542           $ % $
5543           \bfseries \boldmath
5544           $ % $
5545         \else:
5546           \bfseries \boldmath
5547         \fi:
5548       }
5549     }
5550   }
5551   \group_end:
5552   \g_@@_row_style_tl
5553   \ignorespaces
5554 }
```

The following command must *not* be protected.

```
5555 \cs_new:Npn \@@_rounded_from_row:n #1
5556 {
5557   \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
```

In the following code, the “- 1” is *not* a subtraction.

```
5558   { \int_eval:n { #1 } - 1 }
5559   {
5560     \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5561     - \exp_not:n { \int_use:N \c@jCol }
5562   }
5563   { \dim_use:N \l_@@_rounded_corners_dim }
5564 }
```

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_t1`. 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_t1`. 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).

```
5565 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5566 {
```

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.

```
5567 \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`.

```
5568 \str_if_in:nnF { #1 } { !! }
5569 {
5570     \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```
5571     { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5572     }
5573 \int_if_zero:nTF { \l_tmpa_int }
```

First, the case where the color is a *new* color (not in the sequence).

```
5574 {
5575     \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5576     \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ t1 } { #2 }
5577 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5578     { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ t1 } { #2 } }
5579 }
5580 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5581 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

The following command must be used within a `\pgfpicture`.

```
5582 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5583 {
5584     \dim_compare:nNnT { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim }
5585     {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5586 \group_begin:
5587 \pgfsetcornersarced
5588 {
5589     \pgfpoint
5590     { \l_@@_tab_rounded_corners_dim }
5591     { \l_@@_tab_rounded_corners_dim }
5592 }
```

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.

```

5593 \bool_if:NTF \l_@@_hvlines_bool
5594 {
5595     \pgfpathrectanglecorners
5596     {
5597         \pgfpointadd
5598         { \c@_qpoint:n { row-1 } }
5599         { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5600     }
5601     {
5602         \pgfpointadd
5603         {
5604             \c@_qpoint:n
5605             { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5606         }
5607         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5608     }
5609 }
5610 {
5611     \pgfpathrectanglecorners
5612     { \c@_qpoint:n { row-1 } }
5613     {
5614         \pgfpointadd
5615         {
5616             \c@_qpoint:n
5617             { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5618         }
5619         { \pgfpoint \c_zero_dim \arrayrulewidth }
5620     }
5621 }
5622 \pgfusepath { clip }
5623 \group_end:
```

The TeX group was for `\pgfsetcornersarced`.

```

5624 }
5625 }
```

The macro `\c@_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`).

```

5626 \cs_new_protected:Npn \c@_actually_color:
5627 {
5628     \pgfpicture
5629     \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.

```

5630 \c@_clip_with_rounded_corners:
5631 \seq_map_indexed_inline:Nn \g_@@_colors_seq
5632 {
5633     \int_compare:nNnTF { ##1 } = { \c_one_int }
```

```

5634      {
5635          \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5636          \use:c { g_@@_color _ 1 _tl }
5637          \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5638      }
5639      {
5640          \begin { pgfscope }
5641              \@@_color_opacity: ##2
5642              \use:c { g_@@_color _ ##1 _tl }
5643              \tl_gclear:c { g_@@_color _ ##1 _tl }
5644              \pgfusepath { fill }
5645          \end { pgfscope }
5646      }
5647  }
5648  \endpgfpicture
5649 }
```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5650 \cs_new_protected:Npn \@@_color_opacity:
5651 {
5652     \peek_meaning:NTF [
5653         { \@@_color_opacity:w }
5654         { \@@_color_opacity:w [ ] }
5655 }
```

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.

```

5656 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5657 {
5658     \tl_clear:N \l_tmpa_tl
5659     \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.
5660     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillcolor \l_tmpa_tl }
5661     \tl_if_empty:NTF \l_tmpb_tl
5662         { \@declaredcolor }
5663         { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5664 }
```

The following set of keys is used by the command `\@@_color_opacity:w`.

```

5665 \keys_define:nn { nicematrix / color-opacity }
5666 {
5667     opacity .tl_set:N      = \l_tmpa_tl ,
5668     opacity .value_required:n = true
5669 }
```

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

```

5670 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5671 {
5672     \def \l_@@_rows_tl { #1 }
5673     \def \l_@@_cols_tl { #2 }
5674     \@@_cartesian_path:
5675 }
```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5676 \NewDocumentCommand \@@_rowcolor { O { } m m }
5677 {
5678     \tl_if_blank:nF { #2 }
```

```

5679 {
5680     \@@_add_to_colors_seq:en
5681     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5682     { \@@_cartesian_color:nn { #3 } { - } }
5683 }
5684 }
```

Here an example: \@@_columncolor:nn {red!15} {1,3,5-7,10-}

```

5685 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5686 {
5687     \tl_if_blank:nF { #2 }
5688     {
5689         \@@_add_to_colors_seq:en
5690         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5691         { \@@_cartesian_color:nn { - } { #3 } }
5692     }
5693 }
```

Here is an example: \@@_rectanglecolor{red!15}{2-3}{5-6}

```

5694 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5695 {
5696     \tl_if_blank:nF { #2 }
5697     {
5698         \@@_add_to_colors_seq:en
5699         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5700         { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5701     }
5702 }
```

The last argument is the radius of the corners of the rectangle.

```

5703 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5704 {
5705     \tl_if_blank:nF { #2 }
5706     {
5707         \@@_add_to_colors_seq:en
5708         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5709         { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5710     }
5711 }
```

The last argument is the radius of the corners of the rectangle.

```

5712 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5713 {
5714     \@@_cut_on_hyphen:w #1 \q_stop
5715     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5716     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5717     \@@_cut_on_hyphen:w #2 \q_stop
5718     \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5719     \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.

```

5720     \@@_cartesian_path:n { #3 }
5721 }
```

Here is an example: \@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}

```

5722 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5723 {
5724     \clist_map_inline:nn { #3 }
5725     { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
```

```

5727 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5728 {
5729   \int_step_inline:nn { \c@iRow }
5730   {
5731     \int_step_inline:nn { \c@jCol }
5732     {
5733       \int_if_even:nTF { #####1 + ##1 }
5734       { \@@_cellcolor [ #1 ] { #2 } }
5735       { \@@_cellcolor [ #1 ] { #3 } }
5736       { ##1 - #####1 }
5737     }
5738   }
5739 }

```

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”.

```

5740 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5741 {
5742   \@@_rectanglecolor [ #1 ] { #2 }
5743   { 1 - 1 }
5744   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5745 }

5746 \keys_define:nn { nicematrix / rowcolors }
5747 {
5748   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5749   respect-blocks .default:n = true ,
5750   cols .tl_set:N = \l_@@_cols_tl ,
5751   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5752   restart .default:n = true ,
5753   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5754 }

```

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}{red}[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`.

```

5755 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5756 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5757 \group_begin:
5758 \seq_clear_new:N \l_@@_colors_seq
5759 \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5760 \tl_clear_new:N \l_@@_cols_tl
5761 \tl_set:Nn \l_@@_cols_tl { - }
5762 \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).

```

5763 \int_zero_new:N \l_@@_color_int
5764 \int_set_eq:NN \l_@@_color_int \c_one_int
5765 \bool_if:NT \l_@@_respect_blocks_bool
5766 {

```

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`).

```

5767   \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5768   \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5769     { \@@_not_in_exterior_p:nnnnn ##1 }
5770   }
5771   \pgfpicture
5772   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5773   \clist_map_inline:nn { #2 }
5774   {
5775     \tl_set:Nn \l_tmpa_tl { ##1 }
5776     \tl_if_in:NnTF \l_tmpa_tl { - }
5777       { \@@_cut_on_hyphen:w ##1 \q_stop }
5778       { \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.

```

5779   \int_set:Nn \l_tmpa_int \l_tmpa_tl
5780   \int_set:Nn \l_@@_color_int
5781     { \bool_if:NTF \l_@@_rowcolors_restart_bool { 1 } { \l_tmpa_tl } }
5782   \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5783   \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5784   {

```

We will compute in `\l_tmpb_int` the last row of the "block".

```
5785   \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).

```

5786   \bool_if:NT \l_@@_respect_blocks_bool
5787   {
5788     \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5789       { \@@_intersect_our_row_p:nnnnn ####1 }
5790     \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5791   }
5792   \tl_set:Ne \l_@@_rows_tl
5793   { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5794   \tl_set:Ne \l_@@_color_tl
5795   {
5796     \@@_color_index:n
5797     {
5798       \int_mod:nn
5799         { \l_@@_color_int - 1 }
5800         { \seq_count:N \l_@@_colors_seq }
5801       + 1
5802     }
5803   }
5804   \tl_if_empty:NF \l_@@_color_tl
5805   {
5806     \@@_add_to_colors_seq:ee
5807       { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5808       { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5809     }
5810     \int_incr:N \l_@@_color_int
5811     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5812   }
5813 }
5814 \endpgfpicture
5815 \group_end:
5816 }
```

The command `\@@_color_index:n` peekes 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.

```
5817 \cs_new:Npn \@@_color_index:n #1
5818 {
```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```
5819 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5820 { \@@_color_index:n { #1 - 1 } }
5821 { \seq_item:Nn \l_@@_colors_seq { #1 } }
5822 }
```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is a optional argument between square brackets is provided by curryfication.

```
5823 \NewDocumentCommand \@@_rowcolors { O{ } m m m }
5824 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }
```

The braces around #3 and #4 are mandatory.

```
5825 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5826 {
5827     \int_compare:nNnT { #3 } > { \l_tmpb_int }
5828     { \int_set:Nn \l_tmpb_int { #3 } }
5829 }

5830 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5831 {
5832     \int_if_zero:nTF { #4 }
5833     { \prg_return_false: }
5834     {
5835         \int_compare:nNnTF { #2 } > { \c@jCol }
5836         { \prg_return_false: }
5837         { \prg_return_true: }
5838     }
5839 }
```

The following command return `true` when the block intersects the row `\l_tmpa_int`.

```
5840 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5841 {
5842     \int_compare:nNnTF { #1 } > { \l_tmpa_int }
5843     { \prg_return_false: }
5844     {
5845         \int_compare:nNnTF { \l_tmpa_int } > { #3 }
5846         { \prg_return_false: }
5847         { \prg_return_true: }
5848     }
5849 }
```

The following command uses two implicit arguments: `\l_@@_rows_t1` and `\l_@@_cols_t1` 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`).

```
5850 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5851 {
5852     \dim_compare:nNnTF { #1 } = { \c_zero_dim }
5853     {
5854         \bool_if:NTF \l_@@_nocolor_used_bool
5855             { \@@_cartesian_path_normal_ii: }
```

```

5856   {
5857     \clist_if_empty:NTF \l_@@_corners_cells_clist
5858       { \@@_cartesian_path_normal_i:n { #1 } }
5859       { \@@_cartesian_path_normal_ii: }
5860   }
5861 }
5862 { \@@_cartesian_path_normal_i:n { #1 } }
5863 }

```

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.

```

5864 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5865 {
5866   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5867 \clist_map_inline:Nn \l_@@_cols_tl
5868 {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

5869 \def \l_tmpa_tl { ##1 }
5870 \tl_if_in:NnTF \l_tmpa_tl { - }
5871   { \@@_cut_on_hyphen:w ##1 \q_stop }
5872   { \def \l_tmpb_tl { ##1 } }
5873 \tl_if_empty:NTF \l_tmpa_tl
5874   { \def \l_tmpa_tl { 1 } }
5875   {
5876     \str_if_eq:eeT { \l_tmpa_tl } { * }
5877     { \def \l_tmpa_tl { 1 } }
5878   }
5879 \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_col_total_int }
5880   { \@@_error:n { Invalid-col-number } }
5881 \tl_if_empty:NTF \l_tmpb_tl
5882   { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5883   {
5884     \str_if_eq:eeT { \l_tmpb_tl } { * }
5885     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5886   }
5887 \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_col_total_int }
5888   { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5889 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5890 \@@_qpoint:n { col - \l_tmpa_tl }
5891 \int_compare:nNnTF { \l_@@_first_col_int } = { \l_tmpa_tl }
5892   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5893   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5894 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5895 \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.

```

5896 \clist_map_inline:Nn \l_@@_rows_tl
5897 {
5898   \def \l_tmpa_tl { #####1 }
5899   \tl_if_in:NnTF \l_tmpa_tl { - }
5900     { \@@_cut_on_hyphen:w #####1 \q_stop }
5901     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5902   \tl_if_empty:NTF \l_tmpa_tl
5903     { \def \l_tmpa_tl { 1 } }
5904     {
5905       \str_if_eq:eeT { \l_tmpa_tl } { * }
5906       { \def \l_tmpa_tl { 1 } }
5907     }
5908   \tl_if_empty:NTF \l_tmpb_tl

```

```

5909     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5910     {
5911         \str_if_eq:eeT { \l_tmpb_tl } { * }
5912         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5913     }
5914     \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_row_total_int }
5915     { \@@_error:n { Invalid~row~number } }
5916     \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_row_total_int }
5917     { \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`.

```

5918 \cs_if_exist:cF
5919     { @@ _ nocolo r _ \l_tmpa_tl - \l_@@_tmpc_tl }
5920     {
5921         \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5922         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5923         \@@_qpoint:n { row - \l_tmpa_tl }
5924         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5925         \pgfpathrectanglecorners
5926             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5927             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5928     }
5929 }
5930 }
5931 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5932 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5933 {
5934     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5935     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5936 \clist_map_inline:Nn \l_@@_cols_tl
5937 {
5938     \@@_qpoint:n { col - ##1 }
5939     \int_compare:nNnTF { \l_@@_first_col_int } = { ##1 }
5940         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5941         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5942     \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5943     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5944 \clist_map_inline:Nn \l_@@_rows_tl
5945 {
5946     \@@_if_in_corner:nF { #####1 - ##1 }
5947     {
5948         \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5949         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5950         \@@_qpoint:n { row - #####1 }
5951         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5952         \cs_if_exist:cF { @@ _ nocolo r _ #####1 - ##1 }
5953         {
5954             \pgfpathrectanglecorners
5955                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5956                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5957         }
5958     }
5959 }
5960 }

```

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`).

```
5962 \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

```
5963 \cs_new_protected:Npn \@@_cartesian_path_nicolor:n #1
5964 {
5965   \bool_set_true:N \l_@@_nocolor_used_bool
5966   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5967   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
```

We begin the loop over the columns.

```
5968 \clist_map_inline:Nn \l_@@_rows_tl
5969 {
5970   \clist_map_inline:Nn \l_@@_cols_tl
5971   { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
5972 }
5973 }
```

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`.

```
5974 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5975 {
5976   \clist_set_eq:NN \l_tmpa_clist #1
5977   \clist_clear:N #1
5978   \clist_map_inline:Nn \l_tmpa_clist
5979   {
```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```
5980 \def \l_tmpa_tl { ##1 }
5981 \tl_if_in:NnTF \l_tmpa_tl { - }
5982   { \@@_cut_on_hyphen:w ##1 \q_stop }
5983   { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5984 \bool_lazy_or:mnT
5985   { \str_if_eq_p:ee { \l_tmpa_tl } { * } }
5986   { \tl_if_blank_p:o \l_tmpa_tl }
5987   { \def \l_tmpa_tl { 1 } }
5988 \bool_lazy_or:mnT
5989   { \str_if_eq_p:ee { \l_tmpb_t1 } { * } }
5990   { \tl_if_blank_p:o \l_tmpb_t1 }
5991   { \tl_set:No \l_tmpb_t1 { \int_use:N #2 } }
5992 \int_compare:nNnT { \l_tmpb_t1 } > { #2 }
5993   { \tl_set:No \l_tmpb_t1 { \int_use:N #2 } }
5994 \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_t1 }
5995   { \clist_put_right:Nn #1 { #####1 } }
5996 }
5997 }
```

The following command will be linked to `\cellcolor` in the tabular.

```
5998 \NewDocumentCommand \@@_cellcolor_tabular { O{ } m }
5999 {
6000   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6001   {
```

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`).

```
6002   \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
6003     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6004   }
6005   \ignorespaces
6006 }
```

The following command will be linked to `\rowcolor` in the tabular.

```

6007 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
6008 {
6009   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6010   {
6011     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
6012     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6013     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
6014   }
6015   \ignorespaces
6016 }
```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

6017 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
6018 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }
```

The braces around `#2` and `#3` are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

6019 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
6020 { }
```

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`.

```

6021 \seq_gclear:N \g_tmpa_seq
6022 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6023 { \@@_rowlistcolors_tabular:nnnn ##1 }
6024 \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`.

```

6025 \seq_gput_right:Nn \g_@@_rowlistcolors_seq
6026 {
6027   { \int_use:N \c@iRow }
6028   { \exp_not:n { #1 } }
6029   { \exp_not:n { #2 } }
6030   { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
6031 }
6032 \ignorespaces
6033 }
```

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`).

```

6034 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
6035 {
6036   \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).

```

6037 { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
6038 {
6039   \tl_gput_right:Nn \g_@@_pre_code_before_tl
```

```

6040      {
6041        \g_@@_rowlistcolors
6042          [ \exp_not:n { #2 } ]
6043          { #1 - \int_eval:n { \c@iRow - 1 } }
6044          { \exp_not:n { #3 } }
6045          [ \exp_not:n { #4 } ]
6046      }
6047    }
6048  }

```

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.

```

6049 \cs_new_protected:Npn \g_@@_clear_rowlistcolors_seq:
6050  {
6051    \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6052      { \g_@@_rowlistcolors_tabular_ii:nnnn ##1 }
6053    \seq_gclear:N \g_@@_rowlistcolors_seq
6054  }

6055 \cs_new_protected:Npn \g_@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6056  {
6057    \tl_gput_right:Nn \g_@@_pre_code_before_tl
6058      { \g_@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6059  }

```

The first mandatory argument of the command `\g_@@_rowlistcolors` which is written 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.

```

6060 \NewDocumentCommand \columncolor_preamble { O { } m }
6061  {

```

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).

```

6062 \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
6063  {

```

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).

```

6064 \tl_gput_left:Ne \g_@@_pre_code_before_tl
6065  {
6066    \exp_not:N \columncolor [ #1 ]
6067    { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6068  }
6069 }
6070 }

6071 \cs_new_protected:Npn \EmptyColumn:n #1
6072  {
6073    \clist_map_inline:nn { #1 }
6074    {
6075      \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6076        { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6077      \columncolor { nocolor } { ##1 }
6078    }
6079  }

```

```

6080 \cs_new_protected:Npn \@@_EmptyRow:n #1
6081 {
6082     \clist_map_inline:nn { #1 }
6083     {
6084         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6085         { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6086         \rowcolor { nocolor } { ##1 }
6087     }
6088 }

```

22 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumntype` 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`.

```
6089 \cs_set_eq:NN \OnlyMainNiceMatrix \use:
```

Another definition of `\OnlyMainNiceMatrix` will be linked to the command in the environments of `nicematrix`. Here is that definition, called `\@@_OnlyMainNiceMatrix:n`.

```

6090 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6091 {
6092     \int_if_zero:nTF { \l_@@_first_col_int }
6093     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6094     {
6095         \int_if_zero:nTF { \c@jCol }
6096         {
6097             \int_compare:nNnF { \c@iRow } = { -1 }
6098             {
6099                 \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int - 1 }
6100                 { #1 }
6101             }
6102         }
6103         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6104     }
6105 }

```

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.

```

6106 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6107 {
6108     \int_if_zero:nF { \c@iRow }
6109     {
6110         \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
6111         {
6112             \int_compare:nNnT { \c@jCol } > { \c_zero_int }
6113             { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6114         }
6115     }
6116 }

```

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`.

```

6117 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6118 {
6119   \IfPackageLoadedTF { tikz }
6120   {
6121     \IfPackageLoadedTF { booktabs }
6122     {
6123       { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6124     }
6125     { \@@_error:nn { TopRule~without~tikz } { #1 } }
6126   }
6127 \NewExpandableDocumentCommand { \@@_TopRule } { }
6128 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
6129 \cs_new:Npn \@@_TopRule_i:
6130 {
6131   \noalign \bgroup
6132   \peek_meaning:NTF [
6133     { \@@_TopRule_i: }
6134     { \@@_TopRule_i: [ \dim_use:N \heavyrulewidth ] }
6135   }
6136 \NewDocumentCommand \@@_TopRule_i: { o }
6137 {
6138   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6139   {
6140     \@@_hline:n
6141     {
6142       position = \int_eval:n { \c@iRow + 1 } ,
6143       tikz =
6144       {
6145         line-width = #1 ,
6146         yshift = 0.25 \arrayrulewidth ,
6147         shorten< = - 0.5 \arrayrulewidth
6148       } ,
6149       total-width = #1
6150     }
6151   }
6152   \skip_vertical:n { \belowrulesep + #1 }
6153   \egroup
6154 }
6155 \NewExpandableDocumentCommand { \@@_BottomRule } { }
6156 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }
6157 \cs_new:Npn \@@_BottomRule_i:
6158 {
6159   \noalign \bgroup
6160   \peek_meaning:NTF [
6161     { \@@_BottomRule_i: }
6162     { \@@_BottomRule_i: [ \dim_use:N \heavyrulewidth ] }
6163   }
6164 \NewDocumentCommand \@@_BottomRule_i: { o }
6165 {
6166   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6167   {
6168     \@@_hline:n
6169     {
6170       position = \int_eval:n { \c@iRow + 1 } ,
6171       tikz =
6172       {
6173         line-width = #1 ,

```

```

6174     yshift = 0.25 \arrayrulewidth ,
6175     shorten< = - 0.5 \arrayrulewidth
6176   } ,
6177   total-width = #1 ,
6178 }
6179 }
6180 \skip_vertical:N \aboverulesep
6181 \@@_create_row_node_i:
6182 \skip_vertical:n { #1 }
6183 \egroup
6184 }

6185 \NewExpandableDocumentCommand { \@@_MidRule } { }
6186 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }

6187 \cs_new:Npn \@@_MidRule_i:
6188 {
6189   \noalign \bgroup
6190   \peek_meaning:NTF [
6191     { \@@_MidRule_ii: }
6192     { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6193   ]
6194 \NewDocumentCommand \@@_MidRule_ii: { o }
6195 {
6196   \skip_vertical:N \aboverulesep
6197   \@@_create_row_node_i:
6198   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6199   {
6200     \@@_hline:n
6201     {
6202       position = \int_eval:n { \c@iRow + 1 } ,
6203       tikz =
6204       {
6205         line-width = #1 ,
6206         yshift = 0.25 \arrayrulewidth ,
6207         shorten< = - 0.5 \arrayrulewidth
6208       } ,
6209       total-width = #1 ,
6210     }
6211   }
6212   \skip_vertical:n { \belowrulesep + #1 }
6213   \egroup
6214 }

```

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.

```

6215 \keys_define:nn { nicematrix / Rules }
6216 {
6217   position .int_set:N = \l_@@_position_int ,
6218   position .value_required:n = true ,
6219   start .int_set:N = \l_@@_start_int ,
6220   end .code:n =
6221   \bool_lazy_or:nnTF
6222   { \tl_if_empty_p:n { #1 } }
6223   { \str_if_eq_p:ee { #1 } { last } }
6224   { \int_set_eq:NN \l_@@_end_int \c@jCol }
6225   { \int_set:Nn \l_@@_end_int { #1 } }
6226 }

```

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_i:` and `\@@_hline_i::`. Those commands use the following set of keys.

```
6227 \keys_define:nn { nicematrix / RulesBis }
6228 {
6229   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6230   multiplicity .initial:n = 1 ,
6231   dotted .bool_set:N = \l_@@_dotted_bool ,
6232   dotted .initial:n = false ,
6233   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]`.

```
6234   color .code:n =
6235     \@@_set_CArc:n { #1 }
6236     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6237   color .value_required:n = true ,
6238   sep-color .code:n = \@@_set_CDrsc:n { #1 } ,
6239   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.

```
6240 tikz .code:n =
6241   \IfPackageLoadedTF { tikz }
6242   { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6243   { \@@_error:n { tikz-without-tikz } } ,
6244   tikz .value_required:n = true ,
6245   total-width .dim_set:N = \l_@@_rule_width_dim ,
6246   total-width .value_required:n = true ,
6247   width .meta:n = { total-width = #1 } ,
6248   unknown .code:n =
6249     \@@_unknown_key:nn
6250     { nicematrix / RulesBis }
6251     { Unknown-key-for-RulesBis }
6252 }
```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```
6253 \cs_new_protected:Npn \@@_vline:n #1
6254 {
```

The group is for the options.

```
6255 \group_begin:
6256 \int_set_eq:NN \l_@@_end_int \c@iRow
6257 \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```
6258 \int_compare:nNnT { \l_@@_position_int } < { \c@jCol + 2 }
6259   \@@_vline_i:
6260 \group_end:
6261 }
6262 \cs_new_protected:Npn \@@_vline_i:
6263 {
```

\l_tmpa_t1 is the number of row and \l_tmpb_t1 the number of column. When we have found a row corresponding to a rule to draw, we note its number in \l_@@_tmpc_t1.

```

6264     \tl_set:No \l_tmpb_t1 { \int_use:N \l_@@_position_int }
6265     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6266     \l_tmpa_t1
6267     {

```

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.

```

6268     \bool_gset_true:N \g_tmpa_bool
6269     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6270     {
6271         \@@_test_vline_in_block:nnnn ##1
6272     }
6273     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6274     {
6275         \@@_test_vline_in_block:nnnn ##1
6276     }
6277     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6278     {
6279         \@@_test_vline_in_stroken_block:nnnn ##1
6280     }
6281     \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_v: }
6282     \bool_if:NTF \g_tmpa_bool
6283     {
6284         \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.

```

6285     {
6286         \int_set:Nn \l_@@_local_start_int \l_tmpa_t1
6287     }
6288     {
6289         \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6290         {
6291             \int_set:Nn \l_@@_local_end_int { \l_tmpa_t1 - 1 }
6292             \@@_vline_ii:
6293             \int_zero:N \l_@@_local_start_int
6294         }
6295     }
6296
6297     \cs_new_protected:Npn \@@_test_in_corner_v:
6298     {
6299         \int_compare:nNnTF { \l_tmpb_t1 } = { \c_jCol + 1 }
6300         {
6301             \@@_if_in_corner:nT { \l_tmpa_t1 - \int_eval:n { \l_tmpb_t1 - 1 } }
6302             {
6303                 \bool_set_false:N \g_tmpa_bool
6304             }
6305             \@@_if_in_corner:nT { \l_tmpa_t1 - \l_tmpb_t1 }
6306             {
6307                 \int_compare:nNnTF { \l_tmpb_t1 } = { \c_one_int }
6308                 {
6309                     \@@_if_in_corner:nT
6310                     {
6311                         \l_tmpa_t1 - \int_eval:n { \l_tmpb_t1 - 1 }
6312                         {
6313                             \bool_set_false:N \g_tmpa_bool
6314                         }
6315                     }
6316                 }
6317             }
6318         }
6319     }

```

```

6316 \cs_new_protected:Npn \@@_vline_ii:
6317 {
6318   \tl_clear:N \l_@@_tikz_rule_tl
6319   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6320   \bool_if:NTF \l_@@_dotted_bool
6321     { \@@_vline_iv: }
6322     {
6323       \tl_if_empty:NTF \l_@@_tikz_rule_tl
6324         { \@@_vline_iii: }
6325         { \@@_vline_v: }
6326     }
6327 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

6328 \cs_new_protected:Npn \@@_vline_iii:
6329 {
6330   \pgfpicture
6331   \pgfrememberpicturepositiononpagetrue
6332   \pgf@relevantforpicturesizefalse
6333   \Q_Qpoint:n { row - \int_use:N \l_@@_local_start_int }
6334   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6335   \Q_Qpoint:n { col - \int_use:N \l_@@_position_int }
6336   \dim_set:Nn \l_tmpb_dim
6337   {
6338     \pgf@x
6339     - 0.5 \l_@@_rule_width_dim
6340     +
6341     ( \arrayrulewidth * \l_@@_multiplicity_int
6342       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6343   }
6344   \Q_Qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6345   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6346   \bool_lazy_all:nT
6347   {
6348     { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6349     { \cs_if_exist_p:N \CT@drsc@ }
6350     { ! \tl_if_blank_p:o \CT@drsc@ }
6351   }
6352   {
6353     \group_begin:
6354     \CT@drsc@
6355     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6356     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6357     \dim_set:Nn \l_@@_tmpd_dim
6358     {
6359       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6360       * ( \l_@@_multiplicity_int - 1 )
6361     }
6362     \pgfpathrectanglecorners
6363       { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6364       { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6365     \pgfusepath { fill }
6366     \group_end:
6367   }
6368   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6369   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6370   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6371   {
6372     \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6373     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6374     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6375   }
6376 \CT@arc@

```

```

6377   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6378   \pgfsetrectcap
6379   \pgfusepathqstroke
6380   \endpgfpicture
6381 }
```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6382 \cs_new_protected:Npn \@@_vline_iv:
6383 {
6384   \pgfpicture
6385   \pgfrememberpicturepositiononpagetrue
6386   \pgf@relevantforpicturesizefalse
6387   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6388   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6389   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6390   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6391   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6392   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6393   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6394   \CT@arc@
6395   \@@_draw_line:
6396   \endpgfpicture
6397 }
```

The following code is for the case when the user uses the key `tikz`.

```

6398 \cs_new_protected:Npn \@@_vline_v:
6399 {
6400   \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).

```

6401   \CT@arc@
6402   \tl_if_empty:NF \l_@@_rule_color_tl
6403     { \tl_put_right:Nc \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6404   \pgfrememberpicturepositiononpagetrue
6405   \pgf@relevantforpicturesizefalse
6406   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6407   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6408   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6409   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6410   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6411   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6412   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6413   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6414     ( \l_tmpb_dim , \l_tmpa_dim ) --
6415     ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6416   \end{tikzpicture}
6417 }
```

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).

```

6418 \cs_new_protected:Npn \@@_draw_vlines:
6419 {
6420   \int_step_inline:nnn
6421   {
6422     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6423       { 2 }
6424       { 1 }
6425   }
6426   {
6427     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
```

```

6428     { \c@jCol }
6429     { \int_eval:n { \c@jCol + 1 } }
6430   }
6431   {
6432     \str_if_eq:eeF { \l_@@_vlines_clist } { all }
6433     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6434     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6435   }
6436 }

```

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}`.

```

6437 \cs_new_protected:Npn \@@_hline:n #1
6438 {

```

The group is for the options.

```

6439 \group_begin:
6440 \int_set_eq:NN \l_@@_end_int \c@jCol
6441 \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6442 \@@_hline_i:
6443 \group_end:
6444 }
6445 \cs_new_protected:Npn \@@_hline_i:
6446 {

```

`\l_tmpa_t1` is the number of row and `\l_tmpb_t1` the number of column. When we have found a column corresponding to a rule to draw, we note its number in `\l_@@_tmpc_t1`.

```

6447 \tl_set:No \l_tmpa_t1 { \int_use:N \l_@@_position_int }
6448 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6449   \l_tmpb_t1
6450   {

```

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.

```

6451 \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6452 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6453   { \@@_test_hline_in_block:nnnnn ##1 }

6454 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6455   { \@@_test_hline_in_block:nnnnn ##1 }
6456 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6457   { \@@_test_hline_in_stroken_block:nnnn ##1 }
6458 \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_h: }
6459 \bool_if:NTF \g_tmpa_bool
6460   {
6461     \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.

```

6462   { \int_set:Nn \l_@@_local_start_int \l_tmpb_t1 }
6463   }
6464   {
6465     \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6466     {
6467       \int_set:Nn \l_@@_local_end_int { \l_tmpb_t1 - 1 }
6468       \@@_hline_ii:
6469       \int_zero:N \l_@@_local_start_int

```

```

6470         }
6471     }
6472   }
6473 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6474   {
6475     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6476     \@@_hline_ii:
6477   }
6478 }

6479 \cs_new_protected:Npn \@@_test_in_corner_h:
6480 {
6481   \int_compare:nNnTF { \l_tmpa_tl } = { \c_iRow + 1 }
6482   {
6483     \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6484       { \bool_set_false:N \g_tmpa_bool }
6485   }
6486   {
6487     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6488     {
6489       \int_compare:nNnTF { \l_tmpa_tl } = { \c_one_int }
6490         { \bool_set_false:N \g_tmpa_bool }
6491         {
6492           \@@_if_in_corner:nT
6493             { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6494             { \bool_set_false:N \g_tmpa_bool }
6495         }
6496     }
6497   }
6498 }

6499 \cs_new_protected:Npn \@@_hline_ii:
6500 {
6501   \tl_clear:N \l_@@_tikz_rule_tl
6502   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6503   \bool_if:NTF \l_@@_dotted_bool
6504     { \@@_hline_iv: }
6505   {
6506     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6507       { \@@_hline_iii: }
6508       { \@@_hline_v: }
6509   }
6510 }

```

First the case of a standard rule (without the keys `dotted` and `tikz`).

```

6511 \cs_new_protected:Npn \@@_hline_iii:
6512 {
6513   \pgfpicture
6514   \pgfrememberpicturepositiononpagetrue
6515   \pgf@relevantforpicturesizefalse
6516   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6517   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6518   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6519   \dim_set:Nn \l_tmpb_dim
6520   {
6521     \pgf@y
6522     - 0.5 \l_@@_rule_width_dim
6523     +
6524     ( \arrayrulewidth * \l_@@_multiplicity_int
6525       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6526   }

```

```

6527  \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6528  \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6529  \bool_lazy_all:nT
6530  {
6531    { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6532    { \cs_if_exist_p:N \CT@drsc@ }
6533    { ! \tl_if_blank_p:o \CT@drsc@ }
6534  }
6535  {
6536    \group_begin:
6537    \CT@drsc@
6538    \dim_set:Nn \l_@@_tmpd_dim
6539    {
6540      \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6541      * ( \l_@@_multiplicity_int - 1 )
6542    }
6543    \pgfpathrectanglecorners
6544      { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6545      { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6546    \pgfusepathqfill
6547    \group_end:
6548  }
6549  \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6550  \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6551  \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6552  {
6553    \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6554    \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6555    \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6556  }
6557  \CT@arc@
6558  \pgfsetlinewidth { 1.1 \arrayrulewidth }
6559  \pgfsetrectcap
6560  \pgfusepathqstroke
6561  \endpgfpicture
6562 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6563 \cs_new_protected:Npn \@@_hline_iv:
6564  {
6565    \pgfpicture
6566    \pgfrememberpicturepositiononpagetrue
6567    \pgf@relevantforpicturesizefalse
6568    \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6569    \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }

```

```

6570 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6571 \c@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6572 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6573 \int_compare:nNnT { \l_@@_local_start_int } = { \c_one_int }
6574 {
6575   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6576   \bool_if:NF \g_@@_delims_bool
6577     { \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.

```

6578 \tl_if_eq:NnF \g_@@_left_delim_tl (
6579   { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6580 )
6581 \c@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6582 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6583 \int_compare:nNnT { \l_@@_local_end_int } = { \c@jCol }
6584 {
6585   \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6586   \bool_if:NF \g_@@_delims_bool
6587     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6588   \tl_if_eq:NnF \g_@@_right_delim_tl )
6589     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6590 }
6591 \CT@arc@%
6592 \c@_draw_line:
6593 \endpgfpicture
6594 }

```

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`).

```

6595 \cs_new_protected:Npn \c@_hline_v:
6596 {
6597   \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).

```

6598 \CT@arc@%
6599 \tl_if_empty:NF \l_@@_rule_color_tl
6600   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6601 \pgfrememberpicturepositiononpagetrue
6602 \pgf@relevantforpicturesizefalse
6603 \c@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6604 \dim_set_eq:NN \l_tmpa_dim \pgf@x
6605 \c@_qpoint:n { row - \int_use:N \l_@@_position_int }
6606 \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6607 \c@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6608 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6609 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6610 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6611   ( \l_tmpa_dim , \l_tmpb_dim ) --
6612   ( \l_@@_tmpc_dim , \l_tmpb_dim );
6613 \end{tikzpicture}
6614 }

```

The command `\c@_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).

```

6615 \cs_new_protected:Npn \c@_draw_hlines:
6616 {
6617   \int_step_inline:nnn

```

```

6618 { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6619 {
6620   \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6621     { \c@iRow }
6622     { \int_eval:n { \c@iRow + 1 } }
6623   }
6624   {
6625     \str_if_eq:eeF { \l_@@_hlines_clist } { all }
6626       { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6627       { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6628   }
6629 }
```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```
6630 \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.

```

6631 \cs_set:Npn \@@_Hline_i:n #1
6632 {
6633   \peek_remove_spaces:n
6634   {
6635     \peek_meaning:NTF \Hline
6636       { \@@_Hline_ii:nn { #1 + 1 } }
6637       { \@@_Hline_iii:n { #1 } }
6638   }
6639 }
6640 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6641 \cs_set:Npn \@@_Hline_iii:n #1
6642   { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6643 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6644 {
6645   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6646   \skip_vertical:N \l_@@_rule_width_dim
6647   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6648   {
6649     \@@_hline:n
6650     {
6651       multiplicity = #1 ,
6652       position = \int_eval:n { \c@iRow + 1 } ,
6653       total-width = \dim_use:N \l_@@_rule_width_dim ,
6654       #2
6655     }
6656   }
6657   \egroup
6658 }
```

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`).

```

6659 \cs_new_protected:Npn \@@_custom_line:n #1
6660 {
6661   \str_clear_new:N \l_@@_command_str
6662   \str_clear_new:N \l_@@_ccommand_str
6663   \str_clear_new:N \l_@@_letter_str
6664   \tl_clear_new:N \l_@@_other_keys_tl
6665   \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
6666   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
6667   {
6668     \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }
```

We delete the last character which is a space.

```

6669 \str_set:Ne \l_@@_command_str
6670   { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
6671 }
6672 \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
6673 {
6674   \str_set:Ne \l_@@_ccommand_str
6675     { \str_tail:N \l_@@_ccommand_str }
6676   \str_set:Ne \l_@@_ccommand_str
6677     { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
6678 }
```

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.

```

6679 \bool_lazy_all:nTF
6680 {
6681   { \str_if_empty_p:N \l_@@_letter_str }
6682   { \str_if_empty_p:N \l_@@_command_str }
6683   { \str_if_empty_p:N \l_@@_ccommand_str }
6684 }
6685 { \@@_error:n { No-letter-and-no-command } }
6686 { \@@_custom_line_i:o \l_@@_other_keys_tl }
6687 }

6688 \keys_define:nn { nicematrix / custom-line }
6689 {
6690   letter .str_set:N = \l_@@_letter_str ,
6691   letter .value_required:n = true ,
6692   command .str_set:N = \l_@@_command_str ,
6693   command .value_required:n = true ,
6694   ccommand .str_set:N = \l_@@_ccommand_str ,
6695   ccommand .value_required:n = true ,
6696 }
```

6697 \cs_new_protected:Npn \@@_custom_line_i:n #1
6698 {

The following flags will be raised when the keys tikz, dotted and color are used (in the `custom-line`).

```

6699 \bool_set_false:N \l_@@_tikz_rule_bool
6700 \bool_set_false:N \l_@@_dotted_rule_bool
6701 \bool_set_false:N \l_@@_color_bool

6702 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6703 \bool_if:NT \l_@@_tikz_rule_bool
6704 {
6705   \IfPackageLoadedF { tikz }
6706   { \@@_error:n { tikz-in-custom-line-without-tikz } }
6707   \bool_if:NT \l_@@_color_bool
6708   { \@@_error:n { color-in-custom-line-with-tikz } }
6709 }
6710 \bool_if:NT \l_@@_dotted_rule_bool
6711 {
6712   \int_compare:nNnT { \l_@@_multiplicity_int } > { \c_one_int }
6713   { \@@_error:n { key-multiplicity-with-dotted } }
6714 }
6715 \str_if_empty:NF \l_@@_letter_str
6716 {
6717   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6718   { \@@_error:n { Several-letters } }
6719   {
6720     \tl_if_in:NoTF
```

```

6721     \c_@@_forbidden_letters_str
6722     \l_@@_letter_str
6723     { \@@_error:ne { Forbidden-letter } \l_@@_letter_str }
6724     {

```

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.

```

6725         \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6726         { \@@_v_custom_line:nn { #1 } }
6727     }
6728   }
6729 }
6730 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6731 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6732 }
6733 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6734 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6735 \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.

```

6736 \keys_define:nn { nicematrix / custom-line-bis }
6737 {
6738   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6739   multiplicity .initial:n = 1 ,
6740   multiplicity .value_required:n = true ,
6741   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6742   color .value_required:n = true ,
6743   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6744   tikz .value_required:n = true ,
6745   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6746   dotted .value_forbidden:n = true ,
6747   total-width .code:n = { } ,
6748   total-width .value_required:n = true ,
6749   width .code:n = { } ,
6750   width .value_required:n = true ,
6751   sep-color .code:n = { } ,
6752   sep-color .value_required:n = true ,
6753   unknown .code:n =
6754     \@@_unknown_key:nn
6755     { nicematrix / custom-line-bis }
6756     { Unknown~key~for~custom-line }
6757 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6758 \bool_new:N \l_@@_dotted_rule_bool
6759 \bool_new:N \l_@@_tikz_rule_bool
6760 \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`.

```

6761 \keys_define:nn { nicematrix / custom-line-width }
6762 {
6763   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6764   multiplicity .initial:n = 1 ,
6765   multiplicity .value_required:n = true ,
6766   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6767   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }

```

```

6768           \bool_set_true:N \l_@@_total_width_bool ,
6769   total-width .value_required:n = true ,
6770   width .meta:n = { total-width = #1 } ,
6771   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6772 }

```

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).

```

6773 \cs_new_protected:Npn \@@_h_custom_line:n #1
6774 {

```

We use \cs_set:cfn and not \cs_new:cfn because we want a local definition. Moreover, the command must *not* be protected since it begins with \noalign (which is in \Hline).

```

6775 \cs_set_nopar:cfn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6776 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6777 }

```

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).

```

6778 \cs_new_protected:Npn \@@_c_custom_line:n #1
6779 {

```

Here, we need an expandable command since it begins with an \noalign.

```

6780 \exp_args:Nc \NewExpandableDocumentCommand
6781   { nicematrix - \l_@@_ccommand_str }
6782   { O { } m }
6783   {
6784     \noalign
6785     {
6786       \@@_compute_rule_width:n { #1 , ##1 }
6787       \skip_vertical:n { \l_@@_rule_width_dim }
6788       \clist_map_inline:nn
6789         { ##2 }
6790         { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6791     }
6792   }
6793 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6794 }

```

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*.

```

6795 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6796 {
6797   \tl_if_in:nnTF { #2 } { - }
6798   {
6799     \@@_cut_on_hyphen:w #2 \q_stop
6800     \@@_cut_on_hyphen:w #2 - #2 \q_stop
6801   }
6802   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6803   {
6804     \@@_hline:n
6805     {
6806       #1 ,
6807       start = \l_tmpa_tl ,
6808       end = \l_tmpb_tl ,
6809       position = \int_eval:n { \c@iRow + 1 } ,
6810       total-width = \dim_use:N \l_@@_rule_width_dim
6811     }
6812   }

```

```

6812 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6813 {
6814     \bool_set_false:N \l_@@_tikz_rule_bool
6815     \bool_set_false:N \l_@@_total_width_bool
6816     \bool_set_false:N \l_@@_dotted_rule_bool
6817     \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6818     \bool_if:NF \l_@@_total_width_bool
6819     {
6820         \bool_if:NTF \l_@@_dotted_rule_bool
6821             { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6822             {
6823                 \bool_if:NF \l_@@_tikz_rule_bool
6824                 {
6825                     \dim_set:Nn \l_@@_rule_width_dim
6826                     {
6827                         \arrayrulewidth * \l_@@_multiplicity_int
6828                         + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6829                     }
6830                 }
6831             }
6832         }
6833     }

```

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

```

6834 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
6835 {
6836     \str_if_eq:nnTF { #2 } { [ ]
6837         { \@@_v_custom_line_i:nw { #1 } [ ]
6838         { \@@_v_custom_line_ii:nn { #2 } { #1 } }
6839     }
6840     \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
6841     { \@@_v_custom_line:nn { #1 , #2 } }
6842     \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
6843     {
6844         \@@_compute_rule_width:n { #2 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6845     \tl_gput_right:Ne \g_@@_array_preamble_tl
6846         { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6847     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6848     {
6849         \@@_vline:n
6850         {
6851             #2 ,
6852             position = \int_eval:n { \c@jCol + 1 } ,
6853             total-width = \dim_use:N \l_@@_rule_width_dim
6854         }
6855     }
6856     \@@_rec_preamble:n #1
6857 }

6858 \@@_custom_line:n
6859     { letter = : , command = hdottedline , ccommand = cdottedline, dotted }

```

The key `hvlines`

The following command tests whether the current position in the array (given by `\l_tmpa_t1` for the row and `\l_tmpb_t1` 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`.

```

6860 \cs_new_protected:Npn \@@_test_hline_in_block:nnnn #1 #2 #3 #4 #5
6861 {
6862     \int_compare:nNnT { \l_tmpa_t1 } > { #1 }

```

```

6863 {
6864     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6865     {
6866         \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6867         {
6868             \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6869             { \bool_gset_false:N \g_tmpa_bool }
6870         }
6871     }
6872 }
6873 }
```

The same for vertical rules.

```

6874 \cs_new_protected:Npn \@@_test_vline_in_block:nnnn #1 #2 #3 #4 #5
6875 {
6876     \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6877     {
6878         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6879         {
6880             \int_compare:nNnT { \l_tmpb_tl } > { #2 }
6881             {
6882                 \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6883                 { \bool_gset_false:N \g_tmpa_bool }
6884             }
6885         }
6886     }
6887 }
6888 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6889 {
6890     \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6891     {
6892         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6893         {
6894             \int_compare:nNnTF { \l_tmpa_tl } = { #1 }
6895             { \bool_gset_false:N \g_tmpa_bool }
6896             {
6897                 \int_compare:nNnT { \l_tmpa_tl } = { #3 + 1 }
6898                 { \bool_gset_false:N \g_tmpa_bool }
6899             }
6900         }
6901     }
6902 }
6903 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6904 {
6905     \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6906     {
6907         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6908         {
6909             \int_compare:nNnTF { \l_tmpb_tl } = { #2 }
6910             { \bool_gset_false:N \g_tmpa_bool }
6911             {
6912                 \int_compare:nNnT { \l_tmpb_tl } = { #4 + 1 }
6913                 { \bool_gset_false:N \g_tmpa_bool }
6914             }
6915         }
6916     }
6917 }
```

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.

```
6918 \cs_new_protected:Npn \@@_compute_corners:
6919 {
6920     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6921     { \@@_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`).

```
6922 \clist_clear:N \l_@@_corners_cells_clist
6923 \clist_map_inline:Nn \l_@@_corners_clist
6924 {
6925     \str_case:nnF { ##1 }
6926     {
6927         { NW }
6928         { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6929         { NE }
6930         { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6931         { SW }
6932         { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6933         { SE }
6934         { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6935     }
6936     { \@@_error:nn { bad-corner } { ##1 } }
6937 }
```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```
6938 \clist_if_empty:NF \l_@@_corners_cells_clist
6939 {
```

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.

```
6940 \tl_gput_right:Ne \g_@@_aux_tl
6941 {
6942     \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
6943     { \l_@@_corners_cells_clist }
6944 }
6945 }
6946 }

6947 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6948 {
6949     \int_step_inline:nnn { #1 } { #3 }
6950     {
6951         \int_step_inline:nnn { #2 } { #4 }
6952         { \cs_set_nopar:cpn { @@ _ block _ ##1 - #####1 } { } }
6953     }
6954 }

6955 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6956 {
6957     \cs_if_exist:cTF
6958     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6959     { \prg_return_true: }
6960     { \prg_return_false: }
6961 }
```

“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.

```
6962 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6963 {
```

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.

```
6964 \bool_set_false:N \l_tmpa_bool
6965 \int_zero_new:N \l_@@_last_empty_row_int
6966 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6967 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6968 {
6969     \bool_lazy_or:nnTF
6970     {
6971         \cs_if_exist_p:c
6972         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6973     }
6974     { \@@_if_in_block_p:nn { ##1 } { #2 } }
6975     { \bool_set_true:N \l_tmpa_bool }
6976     {
6977         \bool_if:NF \l_tmpa_bool
6978         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6979     }
6980 }
```

Now, you determine the last empty cell in the row of number 1.

```
6981 \bool_set_false:N \l_tmpa_bool
6982 \int_zero_new:N \l_@@_last_empty_column_int
6983 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6984 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6985 {
6986     \bool_lazy_or:nnTF
6987     {
6988         \cs_if_exist_p:c
6989         { \pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6990     }
6991     { \@@_if_in_block_p:nn { #1 } { ##1 } }
6992     { \bool_set_true:N \l_tmpa_bool }
6993     {
6994         \bool_if:NF \l_tmpa_bool
6995         { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6996     }
6997 }
```

Now, we loop over the rows.

```
6998 \int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int }
6999 {
```

We treat the row number `##1` with another loop.

```
7000 \bool_set_false:N \l_tmpa_bool
7001 \int_step_inline:nnnn { #2 } { #4 } { \l_@@_last_empty_column_int }
7002 {
```

```

7003     \bool_lazy_or:nnTF
7004     { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7005     { \@@_if_in_block_p:nn { ##1 } { #####1 } }
7006     { \bool_set_true:N \l_tmpa_bool }
7007     {
7008         \bool_if:NF \l_tmpa_bool
7009         {
7010             \int_set:Nn \l_@@_last_empty_column_int { #####1 }
7011             \clist_put_right:Nn
7012                 \l_@@_corners_cells_clist
7013                 { ##1 - #####1 }
7014             \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7015         }
7016     }
7017 }
7018 }
7019 }
```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

7020 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
7021 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }
```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient:
`\clist_if_in:NeT \l_@@_corners_cells_clist { #1 } ...`

24 The environment {NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```
7022 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

7023 \keys_define:nn { nicematrix / NiceMatrixBlock }
7024 {
7025     auto-columns-width .code:n =
7026     {
7027         \bool_set_true:N \l_@@_block_auto_columns_width_bool
7028         \dim_gzero_new:N \g_@@_max_cell_width_dim
7029         \bool_set_true:N \l_@@_auto_columns_width_bool
7030     }
7031 }
```



```

7032 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
7033 {
7034     \int_gincr:N \g_@@_NiceMatrixBlock_int
7035     \dim_zero:N \l_@@_columns_width_dim
7036     \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
7037     \bool_if:NT \l_@@_block_auto_columns_width_bool
7038     {
7039         \cs_if_exist:cT
7040             { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
7041             {
7042                 \dim_set:Nn \l_@@_columns_width_dim
7043                 {
7044                     \use:c
7045                         { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
7046                 }
7047             }
7048 }
```

```

7048     }
7049 }
```

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`).

```

7050 {
7051   \legacy_if:nTF { measuring@ }
7052   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
7053   {
7054     \bool_if:NT \l_@@_block_auto_columns_width_bool
7055     {
7056       \iow_shipout:Nn \mainaux \ExplSyntaxOn
7057       \iow_shipout:Ne \mainaux
7058       {
7059         \cs_gset:cpn
7060         { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }
```

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.

```

7052   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
7053   {
7054     \bool_if:NT \l_@@_block_auto_columns_width_bool
7055     {
7056       \iow_shipout:Nn \mainaux \ExplSyntaxOn
7057       \iow_shipout:Ne \mainaux
7058       {
7059         \cs_gset:cpn
7060         { @@ _ 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.

```

7061           { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7062           }
7063           \iow_shipout:Nn \mainaux \ExplSyntaxOff
7064         }
7065       }
7066     \ignorespacesafterend
7067   }
```

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).

```

7068 \cs_new_protected:Npn \@@_create_extra_nodes:
7069 {
7070   \bool_if:nTF \l_@@_medium_nodes_bool
7071   {
7072     \bool_if:NTF \l_@@_no_cell_nodes_bool
7073     { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7074     {
7075       \bool_if:NTF \l_@@_large_nodes_bool
7076         \@@_create_medium_and_large_nodes:
7077         \@@_create_medium_nodes:
7078     }
7079   }
7080   {
7081     \bool_if:NT \l_@@_large_nodes_bool
7082     {
7083       \bool_if:NTF \l_@@_no_cell_nodes_bool
7084         { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7085         \@@_create_large_nodes:
7086     }
7087   }
7088 }
```

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`.

```

7089 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7090 {
7091     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7092     {
7093         \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7094         \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7095         \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7096         \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7097     }
7098     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7099     {
7100         \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7101         \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7102         \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7103         \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7104     }

```

We begin the two nested loops over the rows and the columns of the array.

```

7105 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7106 {
7107     \int_step_variable:nnNn
7108         \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.

```

7109 {
7110     \cs_if_exist:cT
7111         { 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`.

```

7112 {
7113     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7114     \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
7115         { \dim_min:vn { l_@@_row_ \@@_i: _min_dim } \pgf@y }
7116     \seq_if_in:NcF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7117     {
7118         \dim_set:cn { l_@@_column_ \@@_j: _min_dim }
7119             { \dim_min:vn { l_@@_column_ \@@_j: _min_dim } \pgf@x }
7120     }

```

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`.

```

7121 \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7122 \dim_set:cn { l_@@_row_ \@@_i: _max_dim }
7123     { \dim_max:vn { l_@@_row_ \@@_i: _max_dim } { \pgf@y } }

```

```

7124     \seq_if_in:Nc \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7125     {
7126         \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
7127         { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } { \pgf@x } }
7128     }
7129 }
7130 }
7131 }
```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7132     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7133     {
7134         \dim_compare:nNnT
7135         { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
7136         {
7137             \@@_qpoint:n { row - \@@_i: - base }
7138             \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
7139             \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
7140         }
7141     }
7142     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7143     {
7144         \dim_compare:nNnT
7145         { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
7146         {
7147             \@@_qpoint:n { col - \@@_j: }
7148             \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
7149             \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
7150         }
7151     }
7152 }
```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7153 \cs_new_protected:Npn \@@_create_medium_nodes:
7154 {
7155     \pgfpicture
7156     \pgfrememberpicturepositiononpagetrue
7157     \pgf@relevantforpicturesizefalse
7158     \@@_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”.

```

7159     \tl_set:Nn \l_@@_suffix_tl { -medium }
7160     \@@_create_nodes:
7161     \endpgfpicture
7162 }
```

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:`.

```

7163 \cs_new_protected:Npn \@@_create_large_nodes:
7164 {
7165     \pgfpicture
7166     \pgfrememberpicturepositiononpagetrue
7167     \pgf@relevantforpicturesizefalse
7168     \@@_computations_for_medium_nodes:
```

¹⁵If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7169 \@@_computations_for_large_nodes:
7170   \tl_set:Nn \l_@@_suffix_tl { - large }
7171   \@@_create_nodes:
7172   \endpgfpicture
7173 }

7174 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7175 {
7176   \pgfpicture
7177     \pgfrememberpicturepositiononpagetrue
7178     \pgf@relevantforpicturesizefalse
7179     \@@_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”.

```

7180   \tl_set:Nn \l_@@_suffix_tl { - medium }
7181   \@@_create_nodes:
7182   \@@_computations_for_large_nodes:
7183     \tl_set:Nn \l_@@_suffix_tl { - large }
7184     \@@_create_nodes:
7185   \endpgfpicture
7186 }

```

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.

```

7187 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7188 {
7189   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7190   \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`.

```

7191   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7192   {
7193     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7194     {
7195       (
7196         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7197         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7198       )
7199       / 2
7200     }
7201     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7202     { l_@@_row _ \@@_i: _ min _ dim }
7203   }
7204   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7205   {
7206     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7207     {
7208       (
7209         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7210         \dim_use:c
7211           { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7212       )
7213       / 2
7214     }
7215     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7216     { l_@@_column _ \@@_j: _ max _ dim }
7217   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7218 \dim_sub:cn
7219   { l_@@_column _ 1 _ min _ dim }
7220   \l_@@_left_margin_dim

```

```

7221   \dim_add:cn
7222     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7223     \l_@@_right_margin_dim
7224 }

```

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).

```

7225 \cs_new_protected:Npn \@@_create_nodes:
7226 {
7227   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7228   {
7229     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7230     {

```

We draw the rectangular node for the cell $(\text{\@@}_i - \text{\@@}_j)$.

```

7231   \@@_pgf_rect_node:nnnn
7232     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7233     { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7234     { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7235     { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
7236     { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7237     \str_if_empty:NF \l_@@_name_str
7238     {
7239       \pgfnodealias
7240         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7241         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7242     }
7243   }
7244 \int_step_inline:nn { \c@iRow }
7245 {
7246   \pgfnodealias
7247     { \@@_env: - ##1 - last \l_@@_suffix_tl }
7248     { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7249   }
7250 \int_step_inline:nn { \c@jCol }
7251 {
7252   \pgfnodealias
7253     { \@@_env: - last - ##1 \l_@@_suffix_tl }
7254     { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7255   }
7256 \pgfnodealias % added 2025-04-05
7257   { \@@_env: - last - last \l_@@_suffix_tl }
7258   { \@@_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 .

```

7260   \seq_map pairwise_function:NNN
7261   \g_@@_multicolumn_cells_seq
7262   \g_@@_multicolumn_sizes_seq
7263   \@@_node_for_multicolumn:nn
7264 }

7265 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7266 {
7267   \cs_set_nopar:Npn \@@_i: { #1 }
7268   \cs_set_nopar:Npn \@@_j: { #2 }
7269 }

```

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”).

```

7270 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7271 {
7272     \@@_extract_coords_values: #1 \q_stop
7273     \@@_pgf_rect_node:nnnn
7274         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7275         { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7276         { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7277         { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7278         { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7279     \str_if_empty:NF \l_@@_name_str
7280     {
7281         \pgfnodealias
7282             { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7283             { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7284     }
7285 }
```

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).

```

7286 \keys_define:nn { nicematrix / Block / FirstPass }
7287 {
7288     j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7289         \bool_set_true:N \l_@@_p_block_bool ,
7290     j .value_forbidden:n = true ,
7291     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7292     l .value_forbidden:n = true ,
7293     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7294     r .value_forbidden:n = true ,
7295     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7296     c .value_forbidden:n = true ,
7297     L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7298     L .value_forbidden:n = true ,
7299     R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7300     R .value_forbidden:n = true ,
7301     C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7302     C .value_forbidden:n = true ,
7303     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7304     t .value_forbidden:n = true ,
7305     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7306     T .value_forbidden:n = true ,
7307     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7308     b .value_forbidden:n = true ,
7309     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7310     B .value_forbidden:n = true ,
7311     m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7312     m .value_forbidden:n = true ,
7313     v-center .meta:n = m ,
7314     p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7315     p .value_forbidden:n = true ,
```

```

7316   color .code:n =
7317     \@@_color:n { #1 }
7318     \tl_set_rescan:Nnn
7319       \l_@@_draw_tl
7320       { \char_set_catcode_other:N ! }
7321       { #1 },
7322     color .value_required:n = true ,
7323     respect_arraystretch .code:n =
7324       \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7325     respect_arraystretch .value_forbidden:n = true ,
7326   }

```

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.

```
7327 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }
```

```

7328 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7329 {

```

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).

```

7330   \tl_if_blank:nTF { #2 }
7331     { \@@_Block_ii:nnnn \c_one_int \c_one_int }
7332     {
7333       \tl_if_in:nnTF { #2 } { - }
7334       {
7335         \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7336         \@@_Block_i_czech:w \@@_Block_i:w
7337         #2 \q_stop
7338       }
7339     {
7340       \@@_error:nn { Bad-argument-for~Block } { #2 }
7341       \@@_Block_ii:nnnn \c_one_int \c_one_int
7342     }
7343   }
7344   { #1 } { #3 } { #4 }
7345   \ignorespaces
7346 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```
7347 \cs_new:Npn \@@_Block_i:w #1-#2 \q_stop { \@@_Block_ii:nnnn { #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`.

```

7348 {
7349   \char_set_catcode_active:N -
7350   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnn { #1 } { #2 } }
7351 }

```

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.

```

7352 \cs_new_protected:Npn \@@_Block_ii:nnnn #1 #2 #3 #4 #5
7353 {

```

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).

```

7354   \bool_lazy_or:nnTF
7355     { \tl_if_blank_p:n { #1 } }
7356     { \str_if_eq_p:ee { * } { #1 } }
7357     { \int_set:Nn \l_tmpa_int { 100 } }
7358     { \int_set:Nn \l_tmpa_int { #1 } }
7359   \bool_lazy_or:nnTF
7360     { \tl_if_blank_p:n { #2 } }
7361     { \str_if_eq_p:ee { * } { #2 } }
7362     { \int_set:Nn \l_tmpb_int { 100 } }
7363     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7364   \int_compare:nNnTF { \l_tmpb_int } = { \c_one_int }
7365   {
7366     \tl_if_empty:NTF \l_@@_hpos_cell_tl
7367       { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7368       { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7369   }
7370   { \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.

```

7371   \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7372   \tl_set:Ne \l_tmpa_tl
7373   {
7374     { \int_use:N \c@iRow }
7375     { \int_use:N \c@jCol }
7376     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7377     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7378   }

```

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).

```

7379   \bool_set_false:N \l_tmpa_bool
7380   \bool_if:NT \l_@@_amp_in_blocks_bool
\tl_if_in:nnT is slightly faster than \str_if_in:nnT.
7381   { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7382   \bool_case:nF
7383   {
7384     \l_tmpa_bool
7385     \l_@@_p_block_bool

```

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.

```

7386   \l_@@_X_bool
7387   { \tl_if_empty_p:n { #5 } }
7388   { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7389   { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7390   }
7391   { \@@_Block_v:eennn }
7392   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7393 }

```

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.

```

7394 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7395 {
7396   \int_gincr:N \g_@@_block_box_int
7397   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7398   {
7399     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7400     {
7401       \@@_actually_diagbox:nnnnnn
7402       { \int_use:N \c@iRow }
7403       { \int_use:N \c@jCol }
7404       { \int_eval:n { \c@iRow + #1 - 1 } }
7405       { \int_eval:n { \c@jCol + #2 - 1 } }
7406       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7407       { \g_@@_row_style_tl \exp_not:n { ##2 } }
7408     }
7409   }
7410   \box_gclear_new:c
7411   { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7412 \hbox_gset:cn
7413 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7414 {

```

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`).

```

7415 \tl_if_empty:NTF \l_@@_color_tl
7416   { \int_compare:nNnT { #2 } = { \c_one_int } { \set@color } }
7417   { \@@_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`.

```

7418 \int_compare:nNnT { #1 } = { \c_one_int }
7419 {
7420   \int_if_zero:nTF { \c@iRow }
7421   {

```

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}$

7422           \cs_set_eq:NN \Block \l_@@_NullBlock:
7423           \l_@@_code_for_first_row_tl
7424       }
7425   {
7426       \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7427       {
7428           \cs_set_eq:NN \Block \l_@@_NullBlock:
7429           \l_@@_code_for_last_row_tl
7430       }
7431   }
7432   \g_@@_row_style_tl
7433 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7434   \@@_reset_arraystretch:
7435   \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7436   #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.).

```

7437   \@@_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}`.

```

7438   \bool_if:NTF \l_@@_tabular_bool
7439   {
7440       \bool_lazy_all:nTF
7441       {
7442           { \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.

```

7443   {
7444       ! \dim_compare_p:nNn
7445           { \l_@@_col_width_dim } < { \c_zero_dim }
7446   }
7447   { ! \g_@@_rotate_bool }
7448 }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7449   {
7450       \use:e
7451       {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7452           \exp_not:N \begin{minipage}
7453               [ \str_lowercase:f \l_@@_vpos_block_str ]
7454               { \l_@@_col_width_dim }
7455               \str_case:on \l_@@_hpos_block_str

```

```

7456      { c \centering r \raggedleft l \raggedright }
7457      }
7458      #5
7459      \end{minipage}
7460  }

```

In the other cases, we use a `{tabular}`.

```

7461  {
7462    \use:e
7463    {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7464      \exp_not:N \begin{tabular}
7465        [ \str_lowercase:f \l_@@_vpos_block_str ]
7466        { @ { } \l_@@_hpos_block_str @ { } }
7467      }
7468      #5
7469      \end{tabular}
7470    }
7471  }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is `false`). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7472  {
7473    $ % $
7474    \use:e
7475    {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7476      \exp_not:N \begin{array}
7477        [ \str_lowercase:f \l_@@_vpos_block_str ]
7478        { @ { } \l_@@_hpos_block_str @ { } }
7479      }
7480      #5
7481      \end{array}
7482      $ % $
7483    }
7484  }

```

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).

```
7485  \bool_if:NT \g_@@_rotate_bool { \c@_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.

```

7486  \int_compare:nNnT { #2 } = { \c_one_int }
7487  {
7488    \dim_gset:Nn \g_@@_blocks_wd_dim
7489    {
7490      \dim_max:nn
7491        { \g_@@_blocks_wd_dim }
7492    {
7493      \box_wd:c
7494        { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7495    }
7496  }
7497 }

```

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.

```

7498  \int_compare:nNnT { #1 } = { \c_one_int }
7499  {

```

```

7500   \bool_lazy_any:nT
7501   {
7502     { \str_if_empty_p:N \l_@@_vpos_block_str }
7503     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
7504     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { b } }
7505   }
7506   { \@@_adjust_blocks_ht_dp: }
7507 }
7508 \seq_gput_right:Ne \g_@@_blocks_seq
7509 {
7510   \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.

```

7511 {
7512   \exp_not:n { #3 } ,
7513   \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7514   \bool_if:NT \g_@@_rotate_bool
7515   {
7516     \bool_if:NTF \g_@@_rotate_c_bool
7517     { m }
7518     {
7519       \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7520       { T }
7521     }
7522   }
7523 }
7524 {
7525   \box_use_drop:c
7526   { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7527 }
7528 }
7529 \bool_set_false:N \g_@@_rotate_c_bool
7530 }

7531 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7532 {
7533   \dim_gset:Nn \g_@@_blocks_ht_dim
7534   {
7535     \dim_max:nn
7536     { \g_@@_blocks_ht_dim }
7537     {
7538       \box_ht:c
7539       { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7540     }
7541   }
7542   \dim_gset:Nn \g_@@_blocks_dp_dim
7543   {
7544     \dim_max:nn
7545     { \g_@@_blocks_dp_dim }
7546     {
7547       \box_dp:c
7548       { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7549     }
7550   }
7551 }

7552 \cs_new:Npn \@@_adjust_hpos_rotate:
7553 {
7554   \bool_if:NT \g_@@_rotate_bool

```

```

7555 {
7556   \str_set:Nn \l_@@_hpos_block_str
7557   {
7558     \bool_if:NTF \g_@@_rotate_c_bool
7559     { c }
7560     {
7561       \str_case:onF \l_@@_vpos_block_str
7562       { b l B l t r T r }
7563       {
7564         \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
7565         { r }
7566         { l }
7567       }
7568     }
7569   }
7570 }
7571 }
7572 \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*.

```

7573 \cs_new_protected:Npn \@@_rotate_box_of_block:
7574 {
7575   \box_grotate:cn
7576   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7577   { 90 }
7578   \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7579   {
7580     \vbox_gset_top:cn
7581     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7582     {
7583       \skip_vertical:n { 0.8 ex }
7584       \box_use:c
7585       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7586     }
7587   }
7588   \bool_if:NT \g_@@_rotate_c_bool
7589   {
7590     \hbox_gset:cn
7591     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7592     {
7593       $ % $
7594       \vcenter
7595       {
7596         \box_use:c
7597         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7598       }
7599       $ % $
7600     }
7601   }
7602 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key p). In that case, the content of the block is *not* composed right away in a box. The composition in a box will be done further, just after the construction of the array (cf. \@@_draw_blocks: and above all \@@_Block_v:nnnnnn).

#1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of key=values pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7603 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7604 {
7605   \seq_gput_right:Ne \g_@@_blocks_seq

```

```

7606   {
7607     \l_tmpa_tl
7608     { \exp_not:n { #3 } }
7609     {
7610       \bool_if:NTF \l_@@_tabular_bool
7611       {
7612         \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7613   \@@_reset_arraystretch:
7614   \exp_not:n
7615   {
7616     \dim_zero:N \extrarowheight
7617     #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).

```

7618   \tag_if_active:T { \tag_stop:n { table } }
7619   \use:e
7620   {
7621     \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7622     { @ { } \l_@@_hpos_block_str @ { } }
7623   }
7624   #5
7625   \end { tabular }
7626 }
7627 \group_end:
7628 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7629   {
7630     \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7631   \@@_reset_arraystretch:
7632   \exp_not:n
7633   {
7634     \dim_zero:N \extrarowheight
7635     #4
7636     $ % $
7637     \use:e
7638     {
7639       \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7640       { @ { } \l_@@_hpos_block_str @ { } }
7641     }
7642     #5
7643     \end { array }
7644     $ % $
7645   }
7646   \group_end:
7647 }
7648 }
7649 }
7650 }
7651 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7652 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7653 {
7654   \seq_gput_right:Ne \g_@@_blocks_seq
7655   {

```

```

7656     \l_tmpa_tl
7657     { \exp_not:n { #3 } }
7658     { { \exp_not:n { #4 #5 } } }
7659   }
7660 }
7661 \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`.

```

7662 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7663 {
7664   \seq_gput_right:Ne \g_@@_blocks_seq
7665   {
7666     \l_tmpa_tl
7667     { \exp_not:n { #3 } }
7668     { \exp_not:n { #4 #5 } }
7669   }
7670 }
7671 \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).

```

7672 \keys_define:nn { nicematrix / Block / SecondPass }
7673 {
7674   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7675   ampersand-in-blocks .default:n = true ,
7676   &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7677 tikz .code:n =
7678   \IfPackageLoadedTF { tikz }
7679   { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7680   { \@@_error:n { tikz-key-without-tikz } } ,
7681 tikz .value_required:n = true ,
7682 fill .code:n =
7683   \tl_set_rescan:Nnn
7684   \l_@@_fill_tl
7685   { \char_set_catcode_other:N ! }
7686   { #1 } ,
7687 fill .value_required:n = true ,

```

In fine, the opacity will be applied by `\pgfsetfillopacity`.

```

7688 opacity .tl_set:N = \l_@@_opacity_tl ,
7689 opacity .value_required:n = true ,
7690 draw .code:n =
7691   \tl_set_rescan:Nnn
7692   \l_@@_draw_tl
7693   { \char_set_catcode_other:N ! }
7694   { #1 } ,
7695 draw .default:n = default ,
7696 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7697 rounded-corners .default:n = 4 pt ,
7698 color .code:n =
7699   \@@_color:n { #1 }
7700   \tl_set_rescan:Nnn
7701   \l_@@_draw_tl
7702   { \char_set_catcode_other:N ! }
7703   { #1 } ,
7704 borders .clist_set:N = \l_@@_borders_clist ,
7705 borders .value_required:n = true ,

```

```

7706 hvlines .meta:n = { vlines , hlines } ,
7707 vlines .bool_set:N = \l_@@_vlines_block_bool,
7708 vlines .default:n = true ,
7709 hlines .bool_set:N = \l_@@_hlines_block_bool,
7710 hlines .default:n = true ,
7711 line-width .dim_set:N = \l_@@_line_width_dim ,
7712 line-width .value_required:n = true ,

```

Some keys have not a property .value_required:n (or similar) because they are in FirstPass.

```

7713 j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7714         \bool_set_true:N \l_@@_p_block_bool ,
7715 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7716 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7717 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7718 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7719         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7720 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7721         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7722 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7723         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7724 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7725 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7726 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7727 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7728 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7729 m .value_forbidden:n = true ,
7730 v-center .meta:n = m ,
7731 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7732 p .value_forbidden:n = true ,
7733 name .tl_set:N = \l_@@_block_name_str , % .str_set:N ?
7734 name .value_required:n = true ,
7735 name .initial:n = ,
7736 respect-arraystretch .code:n =
7737     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7738 respect-arraystretch .value_forbidden:n = true ,
7739 transparent .bool_set:N = \l_@@_transparent_bool ,
7740 transparent .default:n = true ,
7741 transparent .initial:n = false ,
7742 unknown .code:n =
7743     \@@_unknown_key:nn
7744     { nicematrix / Block / SecondPass }
7745     { Unknown~key~for~Block }
7746 }

```

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.

```

7747 \cs_new_protected:Npn \@@_draw_blocks:
7748 {
7749     \bool_if:NTF \c_@@_revtex_bool
7750         { \cs_set_eq:NN \ialign \@@_old_ialign: }
7751         { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7752     \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7753 }
7754 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7755 {

```

The integer \l_@@_last_row_int will be the last row of the block and \l_@@_last_col_int its last column.

```

7756 \int_zero:N \l_@@_last_row_int
7757 \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”).

```

7758 \int_compare:nNnTF { #3 } > { 98 }
7759   { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7760   { \int_set:Nn \l_@@_last_row_int { #3 } }
7761 \int_compare:nNnTF { #4 } > { 98 }
7762   { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7763   { \int_set:Nn \l_@@_last_col_int { #4 } }

7764 \int_compare:nNnTF { \l_@@_last_col_int } > { \g_@@_col_total_int }
7765   {
7766     \bool_lazy_and:nnTF
7767       { \l_@@_preamble_bool }
7768       {
7769         \int_compare_p:n
7770           { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7771       }
7772       {
7773         \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7774         \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7775         \@@_msg_redirect_name:nn { columns-not-used } { none }
7776       }
7777       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7778     }
7779   {
7780     \int_compare:nNnTF { \l_@@_last_row_int } > { \g_@@_row_total_int }
7781       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7782       {
7783         \@@_Block_v:nneenn
7784           { #1 }
7785           { #2 }
7786           { \int_use:N \l_@@_last_row_int }
7787           { \int_use:N \l_@@_last_col_int }
7788           { #5 }
7789           { #6 }
7790         }
7791       }
7792   }

```

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.

```

7793 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7794   {

```

The group is for the keys.

```

7795 \group_begin:
7796 \int_compare:nNnT { #1 } = { #3 }
7797   { \str_set:Nn \l_@@_vpos_block_str { t } }
7798 \keys_set:nn { nicematrix / Block / SecondPass } { #5 }

```

If the content of the block contains `&`, we will have a special treatment (since the cell must be divided in several sub-cells). Remark that `\tl_if_in:nnT` is faster then `\str_if_in:nnT`.

```

7799 \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7800 \bool_lazy_and:nnT
7801   { \l_@@_vlines_block_bool }
7802   { ! \l_@@_ampersand_bool }
7803   {
7804     \tl_gput_right:Ne \g_nicematrix_code_after_tl

```

```

7805      {
7806        \@@_vlines_block:nnn
7807        { \exp_not:n { #5 } }
7808        { #1 - #2 }
7809        { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7810      }
7811    }
7812 \bool_if:NT \l_@@_hlines_block_bool
7813 {
7814   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7815   {
7816     \@@_hlines_block:nnn
7817     { \exp_not:n { #5 } }
7818     { #1 - #2 }
7819     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7820   }
7821 }
7822 \bool_if:NF \l_@@_transparent_bool
7823 {
7824   \bool_lazy_and:nnF { \l_@@_vlines_block_bool } { \l_@@_hlines_block_bool }
7825   {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

7826   \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7827   { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7828 }
7829

7830 \tl_if_empty:NF \l_@@_draw_tl
7831 {
7832   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7833   { \@@_error:n { hlines-with~color } }
7834   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7835   {
7836     \@@_stroke_block:nnn
#5 are the options
7837     { \exp_not:n { #5 } }
7838     { #1 - #2 }
7839     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7840   }
7841   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7842   { { #1 } { #2 } { #3 } { #4 } }
7843 }

7844 \clist_if_empty:NF \l_@@_borders_clist
7845 {
7846   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7847   {
7848     \@@_stroke_borders_block:nnn
7849     { \exp_not:n { #5 } }
7850     { #1 - #2 }
7851     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7852   }
7853 }

7854 \tl_if_empty:NF \l_@@_fill_tl
7855 {
7856   \@@_add_opacity_to_fill:
7857   \tl_gput_right:Ne \g_@@_pre_code_before_tl
7858   {
7859     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7860     { #1 - #2 }

```

```

7861     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7862     { \dim_use:N \l_@@_rounded_corners_dim }
7863   }
7864 }
7865 \seq_if_empty:NF \l_@@_tikz_seq
7866 {
7867   \tl_gput_right:Ne \g_nicematrix_code_before_tl
7868   {
7869     \@@_block_tikz:nnnn
7870     { \seq_use:Nn \l_@@_tikz_seq { , } }
7871     { #1 }
7872     { #2 }
7873     { \int_use:N \l_@@_last_row_int }
7874     { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7875   }
7876 }

7877 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7878 {
7879   \tl_gput_right:Ne \g_@@_pre_code_after_tl
7880   {
7881     \@@_actually_diagbox:nnnnnn
7882     { #1 }
7883     { #2 }
7884     { \int_use:N \l_@@_last_row_int }
7885     { \int_use:N \l_@@_last_col_int }
7886     { \exp_not:n { ##1 } }
7887     { \exp_not:n { ##2 } }
7888   }
7889 }
7890

```

Let's consider the following `\begin{NiceTabular}`. Because of the instruction `!{\hspace{1cm}}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!{\hspace{1cm}}c}
\Block{2-2}{our block} & one & \\
& two & \\
three & four & five \\
six & seven & eight \\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block	one
three	four
six	seven

We highlight the node `1-1-block-short`

our block	one
three	four
six	seven

The construction of the node corresponding to the merged cells.

```

7890 \pgfpicture
7891 \pgfrememberpicturepositiononpagetrue
7892 \pgf@relevantforpicturesizefalse
7893 \@@_qpoint:n { row - #1 }
7894 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7895 \@@_qpoint:n { col - #2 }
7896 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7897 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7898 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7899 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7900 \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:nnnn takes in as arguments the name of the node and the four coordinates of two opposite corner points of the rectangle.

```

7901  \@@_pgf_rect_node:nnnn
7902    { \@@_env: - #1 - #2 - block }
7903    \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7904    \str_if_empty:NF \l_@@_block_name_str
7905    {
7906      \pgfnodealias
7907        { \@@_env: - \l_@@_block_name_str }
7908        { \@@_env: - #1 - #2 - block }
7909      \str_if_empty:NF \l_@@_name_str
7910      {
7911        \pgfnodealias
7912          { \l_@@_name_str - \l_@@_block_name_str }
7913          { \@@_env: - #1 - #2 - block }
7914      }
7915    }

```

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.

```

7916  \bool_if:NF \l_@@_hpos_of_block_cap_bool
7917    {
7918      \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.

```

7919  \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7920    {

```

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.

```

7921  \cs_if_exist:cT
7922    { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7923    {
7924      \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7925      {
7926        \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7927        \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7928      }
7929    }
7930  }

```

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.

```

7931  \dim_compare:nNnT { \l_tmpb_dim } = { \c_max_dim }
7932    {
7933      \@@_qpoint:n { col - #2 }
7934      \dim_set_eq:NN \l_tmpb_dim \pgf@x
7935    }
7936  \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7937  \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7938    {
7939      \cs_if_exist:cT
7940        { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7941        {
7942          \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7943          {
7944            \pgfpointanchor
7945              { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7946              { east }

```

```

7947           \dim_set:Nn \l_@@_tmpd_dim
7948             { \dim_max:nn { \l_@@_tmpd_dim } { \pgf@x } }
7949         }
7950     }
7951   }
7952 \dim_compare:nNnT { \l_@@_tmpd_dim } = { - \c_max_dim }
7953   {
7954     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7955     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7956   }
7957 \@@_pgf_rect_node:nnnn
7958   { \@@_env: - #1 - #2 - block - short }
7959   \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7960 }

```

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.

```

7961 \bool_if:NT \l_@@_medium_nodes_bool
7962   {
7963     \@@_pgf_rect_node:nnn
7964       { \@@_env: - #1 - #2 - block - medium }
7965       { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7966     {
7967       \pgfpointanchor
7968         { \@@_env:
7969           - \int_use:N \l_@@_last_row_int
7970           - \int_use:N \l_@@_last_col_int - medium
7971         }
7972       { south-east }
7973     }
7974   }
7975 \endpgfpicture
7976

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand &.

```

7977 \bool_if:NTF \l_@@_ampersand_bool
7978   {
7979     \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7980     \int_zero_new:N \l_@@_split_int
7981     \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.

```

7982 \int_zero_new:N \l_@@_first_row_int
7983 \int_zero_new:N \l_@@_first_col_int
7984 \int_zero_new:N \l_@@_last_row_int
7985 \int_zero_new:N \l_@@_last_col_int
7986 \int_set:Nn \l_@@_first_row_int { #1 }
7987 \int_set:Nn \l_@@_first_col_int { #2 }
7988 \int_set:Nn \l_@@_last_row_int { #3 }
7989 \int_set:Nn \l_@@_last_col_int { #4 }

7990 \pgfpicture
7991 \pgfrememberpicturepositiononpagetrue
7992 \pgf@relevantforpicturesizefalse
7993 \@@_qpoint:n { row - #1 }
7994 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7995 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7996 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7997 \@@_qpoint:n { col - #2 }
7998 \dim_set_eq:NN \l_tmpa_dim \pgf@x
7999 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8000 \dim_set:Nn \l_tmpb_dim
8001   { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }

```

```

8002 \bool_lazy_or:nnT
8003   { \l_@@_vlines_block_bool }
8004   { \str_if_eq_p:ee { \l_@@_vlines_clist } { all } }
8005   {
8006     \int_step_inline:nn { \l_@@_split_int - 1 }
8007     {
8008       \pgfpathmoveto
8009       {
8010         \pgfpoint
8011         { \l_tmpa_dim + ##1 \l_tmpb_dim }
8012         \l_@@_tmpc_dim
8013       }
8014       \pgfpathlineto
8015       {
8016         \pgfpoint
8017         { \l_tmpa_dim + ##1 \l_tmpb_dim }
8018         \l_@@_tmpd_dim
8019       }
8020       \CT@arc@
8021       \pgfsetlinewidth { 1.1 \arrayrulewidth }
8022       \pgfsetrectcap
8023       \pgfusepathqstroke
8024     }
8025   }
8026 \cs_set_eq:NN \cellcolor \@@_subcellcolor
8027 \int_zero_new:N \l_@@_split_i_int
8028 \str_if_eq:eeTF { \l_@@_vpos_block_str } { T }
8029   {
8030     \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
8031     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8032   }
8033   {
8034     \str_if_eq:eeTF { \l_@@_vpos_block_str } { B }
8035     {
8036       \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
8037       \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8038     }
8039     {
8040       \bool_lazy_or:nnTF
8041         { \int_compare_p:nNn { #1 } = { #3 } }
8042         { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
8043         {
8044           \@@_qpoint:n { row - #1 - base }
8045           \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8046         }
8047         {
8048           \str_if_eq:eeTF { \l_@@_vpos_block_str } { b }
8049             {
8050               \@@_qpoint:n { row - #3 - base }
8051               \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8052             }
8053             {
8054               \@@_qpoint:n { #1 - #2 - block }
8055               \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8056             }
8057           }
8058         }
8059       }
8060     \int_step_inline:nn { \l_@@_split_int }
8061     {
8062       \group_begin:

```

The counter `\l_@@_split_i_int` is only for the command `\@@_subcellcolor`.

```

8063 \int_set:Nn \l_@@_split_i_int { ##1 }
8064 \dim_set:Nn \col@sep
8065   { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
8066 \pgftransformshift
8067   {
8068     \pgfpoint
8069     {
8070       \l_tmpa_dim + ##1 \l_tmpb_dim -
8071       \str_case:on \l_@@_hpos_block_str
8072       {
8073         l { \l_tmpb_dim + \col@sep}
8074         c { 0.5 \l_tmpb_dim }
8075         r { \col@sep }
8076       }
8077     }
8078   { \l_@@_tmpc_dim }
8079 }
8080 \pgfset { inner~sep = \c_zero_dim }
8081 \pgfnode
8082   { rectangle }
8083 {
8084   \str_if_eq:eeTF { \l_@@_vpos_block_str } { T }
8085   {
8086     \str_case:on \l_@@_hpos_block_str
8087     {
8088       l { north-west }
8089       c { north }
8090       r { north-east }
8091     }
8092   }
8093 {
8094   \str_if_eq:eeTF { \l_@@_vpos_block_str } { B }
8095   {
8096     \str_case:on \l_@@_hpos_block_str
8097     {
8098       l { south-west }
8099       c { south }
8100       r { south-east }
8101     }
8102   }
8103 {
8104   \bool_lazy_any:nTF
8105   {
8106     { \int_compare_p:nNn { #1 } = { #3 } }
8107     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
8108     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { b } }
8109   }
8110 {
8111   \str_case:on \l_@@_hpos_block_str
8112   {
8113     l { base-west }
8114     c { base }
8115     r { base-east }
8116   }
8117 }
8118 {
8119   \str_case:on \l_@@_hpos_block_str
8120   {
8121     l { west }
8122     c { center }
8123     r { east }
8124   }
8125 }

```

```

8126         }
8127     }
8128     {
8129       \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8130     \group_end:
8131   }
8132 \endpgfpicture
8133 }

```

Now the case where there is no ampersand & in the content of the block.

```

8134 {
8135   \bool_if:NTF \l_@@_p_block_bool
8136   {

```

When the final user has used the key p, we have to compute the width.

```

8137 \pgfpicture
8138   \pgfrememberpicturepositiononpagetrue
8139   \pgf@relevantforpicturesizefalse
8140   \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8141   {
8142     \@@_qpoint:n { col - #2 }
8143     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8144     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8145   }
8146   {
8147     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8148     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8149     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8150   }
8151   \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8152 \endpgfpicture
8153 \hbox_set:Nn \l_@@_cell_box
8154 {
8155   \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8156   { \g_tmpb_dim }
8157   \str_case:on \l_@@_hpos_block_str
8158   { c \centering r \raggedleft l \raggedright j { } }
8159   #6
8160   \end { minipage }
8161 }
8162 {
8163   \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
8164 \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.

```

8165 \pgfpicture
8166 \pgfrememberpicturepositiononpagetrue
8167 \pgf@relevantforpicturesizefalse
8168 \bool_lazy_any:nTF
8169 {
8170   { \str_if_empty_p:N \l_@@_vpos_block_str }
8171   { \str_if_eq_p:ee { \l_@@_vpos_block_str } { c } }
8172   { \str_if_eq_p:ee { \l_@@_vpos_block_str } { T } }
8173   { \str_if_eq_p:ee { \l_@@_vpos_block_str } { B } }
8174 }

8175 {

```

If we are in the “first column”, we must put the block as if it was with the key r.

```

8176   \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`.

```

8177 \bool_if:nT \g_@@_last_col_found_bool
8178 {
8179     \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
8180     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8181 }
```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8182 \tl_set:Nn \l_tmpa_tl
8183 {
8184     \str_case:on \l_@@_vpos_block_str
8185 }
```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8186 { } {
8187     \str_case:on \l_@@_hpos_block_str
8188     {
8189         c { center }
8190         l { west }
8191         r { east }
8192         j { center }
8193     }
8194 }
8195 c {
8196     \str_case:on \l_@@_hpos_block_str
8197     {
8198         c { center }
8199         l { west }
8200         r { east }
8201         j { center }
8202     }
8203 }
8204 T {
8205     \str_case:on \l_@@_hpos_block_str
8206     {
8207         c { north }
8208         l { north-west }
8209         r { north-east }
8210         j { north }
8211     }
8212 }
8213 }
8214 B {
8215     \str_case:on \l_@@_hpos_block_str
8216     {
8217         c { south }
8218         l { south-west }
8219         r { south-east }
8220         j { south }
8221     }
8222 }
8223 }
8224 }
8225 }
8226 }
8227 \pgftransformshift
8228 {
8229     \pgfpointanchor
8230     {
8231         \@@_env: - #1 - #2 - block
8232         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8233     }
8234 { \l_tmpa_tl }
```

```

8235     }
8236     \pgfset { inner~sep = \c_zero_dim }
8237     \pgfnode
8238         { rectangle }
8239         { \l_tmpa_tl }
8240         { \box_use_drop:N \l_@@_cell_box } { } { }
8241     }

```

End of the case when $\l_@@_vpos_block_{str}$ is equal to c, T or B. Now, the other cases.

```

8242 {
8243     \pgfextracty \l_tmpa_dim
8244     {
8245         \@@_qpoint:n
8246         {
8247             \row - \str_if_eq:eeTF { \l_@@_vpos_block_{str} } { b } { #3 } { #1 }
8248             - base
8249         }
8250     }
8251     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in $\pgf@x$) the x -value of the center of the block.

```

8252 \pgfpointanchor
8253 {
8254     \@@_env: - #1 - #2 - block
8255     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8256 }
8257 {
8258     \str_case:on \l_@@_hpos_block_{str}
8259     {
8260         c { center }
8261         l { west }
8262         r { east }
8263         j { center }
8264     }
8265 }

```

We put the label of the block which has been composed in $\l_@@_cell_box$.

```

8266 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8267 \pgfset { inner~sep = \c_zero_dim }
8268 \pgfnode
8269     { rectangle }
8270     {
8271         \str_case:on \l_@@_hpos_block_{str}
8272         {
8273             c { base }
8274             l { base-west }
8275             r { base-east }
8276             j { base }
8277         }
8278     }
8279     { \box_use_drop:N \l_@@_cell_box } { } { }
8280 }
8281 \endpgfpicture
8282 }
8283 \group_end:
8284 }
8285 \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).

```

8286 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8287 {

```

```

8288 \tl_if_empty:NF \l_@@_opacity_tl
8289 {
8290     \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8291     {
8292         \tl_set:Ne \l_@@_fill_tl
8293         {
8294             [ opacity = \l_@@_opacity_tl ,
8295             \tl_tail:o \l_@@_fill_tl
8296         }
8297     }
8298     {
8299         \tl_set:Ne \l_@@_fill_tl
8300         { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8301     }
8302 }
8303 }
```

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).

```

8304 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8305 {
8306     \group_begin:
8307     \tl_clear:N \l_@@_draw_tl
8308     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8309     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8310     \pgfpicture
8311     \pgfrememberpicturepositiononpagetrue
8312     \pgf@relevantforpicturesizefalse
8313     \tl_if_empty:NF \l_@@_draw_tl
8314     {
```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8315 \tl_if_eq:NnTF \l_@@_draw_tl { default }
8316     { \CT@arc@ }
8317     { \@@_color:o \l_@@_draw_tl }
8318 }
8319 \pgfsetcornersarced
8320 {
8321     \pgfpoint
8322     { \l_@@_rounded_corners_dim }
8323     { \l_@@_rounded_corners_dim }
8324 }
8325 \@@_cut_on_hyphen:w #2 \q_stop
8326 \int_compare:nNnF { \l_tmpa_tl } > { \c@iRow }
8327 {
8328     \int_compare:nNnF { \l_tmpb_tl } > { \c@jCol }
8329     {
8330         \@@_qpoint:n { row - \l_tmpa_tl }
8331         \dim_set_eq:NN \l_tmpb_dim \pgf@y
8332         \@@_qpoint:n { col - \l_tmpb_tl }
8333         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8334         \@@_cut_on_hyphen:w #3 \q_stop
8335         \int_compare:nNnT { \l_tmpa_tl } > { \c@iRow }
8336             { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8337         \int_compare:nNnT { \l_tmpb_tl } > { \c@jCol }
8338             { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8339             \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8340             \dim_set_eq:NN \l_tmpa_dim \pgf@y
8341             \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8342             \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8343             \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
```

```

8344     \pgfpathrectanglecorners
8345         { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8346         { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8347     \dim_compare:nNnTF { \l_@@_rounded_corners_dim } = { \c_zero_dim }
8348         { \pgfusepathqstroke }
8349         { \pgfusepath { stroke } }
8350     }
8351   }
8352 \endpgfpicture
8353 \group_end:
8354 }
```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8355 \keys_define:nn { nicematrix / BlockStroke }
8356 {
8357   color .tl_set:N = \l_@@_draw_tl ,
8358   draw .code:n =
8359     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8360   draw .default:n = default ,
8361   line-width .dim_set:N = \l_@@_line_width_dim ,
8362   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8363   rounded-corners .default:n = 4 pt
8364 }
```

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).

```

8365 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8366 {
8367   \group_begin:
8368   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8369   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8370   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8371   \@@_cut_on_hyphen:w #2 \q_stop
8372   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8373   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8374   \@@_cut_on_hyphen:w #3 \q_stop
8375   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8376   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8377   \int_step_inline:nnn { \l_@@_tmpd_tl } { \l_tmpb_tl }
8378   {
8379     \use:e
8380     {
8381       \@@_vline:n
8382       {
8383         position = ##1 ,
8384         start = \l_@@_tmpc_tl ,
8385         end = \int_eval:n { \l_tmpa_tl - 1 } ,
8386         total-width = \dim_use:N \l_@@_line_width_dim
8387       }
8388     }
8389   }
8390   \group_end:
8391 }
8392 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8393 {
8394   \group_begin:
8395   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8396   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8397   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8398   \@@_cut_on_hyphen:w #2 \q_stop
8399   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8400   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
```

```

8401 \@@_cut_on_hyphen:w #3 \q_stop
8402 \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8403 \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8404 \int_step_inline:nnn { \l_@@_tmpc_tl } { \l_tmpa_tl }
8405 {
8406   \use:e
8407   {
8408     \@@_hline:n
8409     {
8410       position = ##1 ,
8411       start = \l_@@_tmpd_tl ,
8412       end = \int_eval:n { \l_tmpb_tl - 1 } ,
8413       total-width = \dim_use:N \l_@@_line_width_dim
8414     }
8415   }
8416 }
8417 \group_end:
8418 }
```

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).

```

8419 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8420 {
8421   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8422   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8423   \dim_compare:nNnTF { \l_@@_rounded_corners_dim } > { \c_zero_dim }
8424   { \@@_error:n { borders-forbidden } }
8425   {
8426     \tl_clear_new:N \l_@@_borders_tikz_tl
8427     \keys_set:no
8428       { nicematrix / OnlyForTikzInBorders }
8429       \l_@@_borders_clist
8430     \@@_cut_on_hyphen:w #2 \q_stop
8431     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8432     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8433     \@@_cut_on_hyphen:w #3 \q_stop
8434     \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8435     \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8436     \@@_stroke_borders_block_i:
8437   }
8438 }
8439 \hook_gput_code:nnn { begindocument } { . }
8440 {
8441   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8442   {
8443     \c_@@_pgfornikzpicture_tl
8444     \@@_stroke_borders_block_ii:
8445     \c_@@_endpgfornikzpicture_tl
8446   }
8447 }
8448 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8449 {
8450   \pgfrememberpicturepositiononpagetrue
8451   \pgf@relevantforpicturesizefalse
8452   \CT@arc@
8453   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8454   \clist_if_in:NnT \l_@@_borders_clist { right }
8455   { \@@_stroke_vertical:n \l_tmpb_tl }
8456   \clist_if_in:NnT \l_@@_borders_clist { left }
8457   { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8458   \clist_if_in:NnT \l_@@_borders_clist { bottom }
```

```

8459 { \@@_stroke_horizontal:n \l_tmpa_tl }
8460 \clist_if_in:NnT \l_@@_borders_clist { top }
8461 { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8462 }
8463 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8464 {
8465 tikz .code:n =
8466 \cs_if_exist:NTF \tikzpicture
8467 { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8468 { \@@_error:n { tikz-in-borders-without-tikz } },
8469 tikz .value_required:n = true ,
8470 top .code:n = ,
8471 bottom .code:n = ,
8472 left .code:n = ,
8473 right .code:n = ,
8474 unknown .code:n = \@@_error:n { bad-border }
8475 }

```

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).

```

8476 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8477 {
8478 \@@_qpoint:n \l_@@_tmpc_tl
8479 \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8480 \@@_qpoint:n \l_tmpa_tl
8481 \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8482 \@@_qpoint:n { #1 }
8483 \tl_if_empty:NTF \l_@@_borders_tikz_tl
8484 {
8485 \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8486 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8487 \pgfusepathqstroke
8488 }
8489 {
8490 \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8491 ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8492 }
8493 }

```

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).

```

8494 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8495 {
8496 \@@_qpoint:n \l_@@_tmpd_tl
8497 \clist_if_in:NnTF \l_@@_borders_clist { left }
8498 { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8499 { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8500 \@@_qpoint:n \l_tmpb_tl
8501 \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8502 \@@_qpoint:n { #1 }
8503 \tl_if_empty:NTF \l_@@_borders_tikz_tl
8504 {
8505 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8506 \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8507 \pgfusepathqstroke
8508 }
8509 {
8510 \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8511 ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8512 }
8513 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

8514 \keys_define:nn { nicematrix / BlockBorders }
8515 {
8516   borders .clist_set:N = \l_@@_borders_clist ,
8517   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8518   rounded-corners .default:n = 4 pt ,
8519   line-width .dim_set:N = \l_@@_line_width_dim
8520 }
```

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.

```

8521 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8522 {
8523   \begin{tikzpicture}
8524     \@@_clip_with_rounded_corners:
```

We use `clist_map_inline:nn` because #5 is a list of lists.

```

8525   \clist_map_inline:nn { #1 }
8526   {
```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8527   \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8528   \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8529   (
8530     [
8531       xshift = \dim_use:N \l_@@_offset_dim ,
8532       yshift = - \dim_use:N \l_@@_offset_dim
8533     ]
8534     #2 -| #3
8535   )
8536   rectangle
8537   (
8538     [
8539       xshift = - \dim_use:N \l_@@_offset_dim ,
8540       yshift = \dim_use:N \l_@@_offset_dim
8541     ]
8542     \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8543   );
8544 }
8545 \end{tikzpicture}
8546 }
8547 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

8548 \keys_define:nn { nicematrix / SpecialOffset }
8549   { 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.

```

8550 \cs_new_protected:Npn \@@_NullBlock:
8551   { \@@_collect_options:n { \@@_NullBlock_i: } }
8552 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8553   { }
```

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.

```
8554 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
8555 {
8556   \tl_gput_right:Nn \g_@@_pre_code_before_tl
8557 }
```

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).

```
8558   \@@_subcellcolor:nnnnnnn
8559   {
8560     \tl_if_blank:nTF { #1 }
8561     { { \exp_not:n { #2 } } }
8562     { [ #1 ] { \exp_not:n { #2 } } }
8563   }
8564   { \int_use:N \l_@@_first_row_int } % first row of the block
8565   { \int_use:N \l_@@_first_col_int } % first column of the block
8566   { \int_use:N \l_@@_last_row_int } % last row of the block
8567   { \int_use:N \l_@@_last_col_int } % last column of the block
8568   { \int_use:N \l_@@_split_int }
8569   { \int_use:N \l_@@_split_i_int }
8570 }
8571 \ignorespaces
8572 }

8573 \cs_new_protected:Npn \@@_subcellcolor:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8574 {
8575   \@@_color_opacity: #1
8576   \pgfpicture
8577   \pgf@relevantforpicturesizefalse
8578   \@@_qpoint:n { col - #3 }
8579   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8580   \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
8581   \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
8582   \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
8583   \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
8584   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8585   \@@_qpoint:n { row - #2 }
8586   \pgfpathrectanglecorners
8587   { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } { \l_@@_tmpc_dim } }
8588   { \pgfpoint { \l_tmpb_dim } { \pgf@y } }
8589   \pgfusepathqfill
8590   \endpgfpicture
8591 }
```

27 How to draw the dotted lines transparently

```
8592 \cs_set_protected:Npn \@@_renew_matrix:
8593 {
8594   \RenewDocumentEnvironment { pmatrix } { }
8595   { \pNiceMatrix }
8596   { \endpNiceMatrix }
8597   \RenewDocumentEnvironment { vmatrix } { }
8598   { \vNiceMatrix }
8599   { \endvNiceMatrix }
8600   \RenewDocumentEnvironment { Vmatrix } { }
8601   { \VNiceMatrix }
8602   { \endVNiceMatrix }
8603   \RenewDocumentEnvironment { bmatrix } { }
8604   { \bNiceMatrix }
8605   { \endbNiceMatrix }
```

```

8606 \RenewDocumentEnvironment { Bmatrix } { }
8607   { \BNiceMatrix }
8608   { \endBNiceMatrix }
8609 }

```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8610 \keys_define:nn { nicematrix / Auto }
8611   {
8612     columns-type .tl_set:N = \l_@@_columns_type_tl ,
8613     columns-type .value_required:n = true ,
8614     l .meta:n = { columns-type = l } ,
8615     r .meta:n = { columns-type = r } ,
8616     c .meta:n = { columns-type = c } ,
8617     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8618     delimiters / color .value_required:n = true ,
8619     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8620     delimiters / max-width .default:n = true ,
8621     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8622     delimiters .value_required:n = true ,
8623     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8624     rounded-corners .default:n = 4 pt
8625   }
8626 \NewDocumentCommand \AutoNiceMatrixWithDelims
8627   { m m O { } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8628   { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
8629 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8630   {

```

The group is for the protection of the keys.

```

8631 \group_begin:
8632 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8633 \use:e
8634   {
8635     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8636     { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8637     [ \exp_not:o \l_tmpa_tl ]
8638   }
8639 \int_if_zero:nT { \l_@@_first_row_int }
8640   {
8641     \int_if_zero:nT { \l_@@_first_col_int } { & }
8642     \prg_replicate:nn { #4 - 1 } { & }
8643     \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8644   }
8645 \prg_replicate:nn { #3 }
8646   {
8647     \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`).

```

8648 \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8649 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8650   }
8651 \int_compare:nNnT { \l_@@_last_row_int } > { -2 } \\
8652   {
8653     \int_if_zero:nT { \l_@@_first_col_int } { & }
8654     \prg_replicate:nn { #4 - 1 } { & }
8655     \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8656   }

```

```

8657     \end { NiceArrayWithDelims }
8658     \group_end:
8659   }
8660 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
8661 {
8662   \cs_set_protected:cpx { #1 AutoNiceMatrix }
8663   {
8664     \bool_gset_true:N \g_@@_delims_bool
8665     \str_gset:Nn \g_@@_name_env_str { #1 AutoNiceMatrix }
8666     \AutoNiceMatrixWithDelims { #2 } { #3 }
8667   }
8668 }
```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8669 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8670 {
8671   \group_begin:
8672   \bool_gset_false:N \g_@@_delims_bool
8673   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8674   \group_end:
8675 }
```

29 The redefinition of the command `\dotfill`

```

8676 \cs_set_eq:NN \@@_old_dotfill: \dotfill
8677 \cs_new_protected:Npn \@@_dotfill:
8678 {
```

First, we insert `\@@_old_dotfill` (which is the saved version of `\dotfill`) in case of use of `\dotfill` “internally” in the cell (e.g. `\hbox to 1cm {\dotfill}`).

```

8679   \@@_old_dotfill:
8680   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8681 }
```

Now, if the box if 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`.

```

8682 \cs_new_protected:Npn \@@_dotfill_i:
8683 {
8684   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = { \c_zero_dim }
8685   { \@@_old_dotfill: }
8686 }
```

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.

```

8687 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8688 {
8689   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8690   {
8691     \@@_actually_diagbox:nnnnnn
8692     { \int_use:N \c@iRow }
8693     { \int_use:N \c@jCol }
8694     { \int_use:N \c@iRow }
8695     { \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.

```
8696     { \g_@@_row_style_tl \exp_not:n { #1 } }
8697     { \g_@@_row_style_tl \exp_not:n { #2 } }
8698 }
```

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.

```
8699 \seq_gput_right:Nne \g_@@_pos_of_blocks_seq
8700 {
8701     { \int_use:N \c@iRow }
8702     { \int_use:N \c@jCol }
8703     { \int_use:N \c@iRow }
8704     { \int_use:N \c@jCol }
```

The last argument is for the name of the block.

```
8705     { }
8706 }
8707 }
```

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.

```
8708 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8709 {
8710     \pgfpicture
8711     \pgf@relevantforpicturesizefalse
8712     \pgfrememberpicturepositiononpagetrue
8713     \@@_qpoint:n { row - #1 }
8714     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8715     \@@_qpoint:n { col - #2 }
8716     \dim_set_eq:NN \l_tmpb_dim \pgf@x
8717     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8718     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8719     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8720     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8721     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8722     \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8723 }
```

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.

```
8724     \CT@arc@
8725     \pgfsetroundcap
8726     \pgfusepathqstroke
8727 }
8728 \pgfset { inner-sep = 1 pt }
8729 \pgfscope
8730 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8731 \pgfnode { rectangle } { south-west }
8732 {
8733     \begin { minipage } { 20 cm }
```

The \scan_stop: avoids an error in math mode when the argument #5 is empty.

```
8734     \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8735     \end { minipage }
8736 }
8737 {
8738 }
```

```

8739 \endpgfscope
8740 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8741 \pgfnode { rectangle } { north-east }
8742 {
8743   \begin { minipage } { 20 cm }
8744   \raggedleft
8745   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8746   \end { minipage }
8747 }
8748 {
8749 {
8750 \endpgfpicture
8751 }

```

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`.

```
8752 \cs_new:Npn \@@_CodeAfter: { \omit \@@_CodeAfter_i:n }
```

However, in each cell of the environment, the command `\CodeAfter` will be linked to the following command `\@@_CodeAfter_i:n` which begins with `\\\`.

```
8753 \cs_new_protected:Npn \@@_CodeAfter_i: { \\ \omit \@@_CodeAfter_i:n }
```

We have to catch everything until the end of the current environment (of `nicematrix`). First, we go until the next command `\end`.

```

8754 \cs_new_protected:Npn \@@_CodeAfter_i:n #1 \end
8755 {
8756   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8757   \@@_CodeAfter_iv:n
8758 }
```

We catch the argument of the command `\end` (in `#1`).

```
8759 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8760 {
```

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.

```
8761 \str_if_eq:eeTF { \currenvir } { #1 }
8762 { \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`.

```

8763 {
8764   \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8765   \@@_CodeAfter_i:n
8766 }
8767 }
```

32 The delimiters in the preamble

The command `\@@_delimiter:n` 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:n` in the `\g_@@_pre_code_after_t1` (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.

```
8768 \cs_new_protected:Npn \@@_delimiter:n #1 #2 #3
8769 {
8770   \pgfpicture
8771   \pgfrememberpicturepositiononpagetrue
8772   \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.

```
8773 \@@_qpoint:n { row - 1 }
8774 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8775 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8776 \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).

```
8777 \bool_if:nTF { #3 }
8778   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8779   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8780 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8781 {
8782   \cs_if_exist:cT
8783     { \pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8784   {
8785     \pgfpointanchor
8786       { \@@_env: - ##1 - #2 }
8787       { \bool_if:nTF { #3 } { west } { east } }
8788   \dim_set:Nn \l_tmpa_dim
8789   {
8790     \bool_if:nTF { #3 }
8791       { \dim_min:nn }
8792       { \dim_max:nn }
8793     \l_tmpa_dim
8794     { \pgf@x }
8795   }
8796 }
8797 }
```

Now we can put the delimiter with a node of PGF.

```
8798 \pgfset { inner_sep = \c_zero_dim }
8799 \dim_zero:N \nulldelimterspace
8800 \pgftransformshift
8801 {
8802   \pgfpoint
8803     { \l_tmpa_dim }
8804     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8805   }
8806 \pgfnode
8807   { rectangle }
8808   { \bool_if:nTF { #3 } { east } { west } }
8809 {
```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8810 \nullfont
8811 $ % $
8812 \@@_color:o \l_@@_delimiters_color_tl
8813 \bool_if:nTF { #3 } { \left #1 } { \left . }
8814 \vcenter
8815 {
8816   \nullfont
8817   \hrule \Oheight
8818     \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8819     \Odepth \c_zero_dim
8820     \Owidth \c_zero_dim
8821   }
8822   \bool_if:nTF { #3 } { \right . } { \right #1 }
8823 $ % $
8824 }
8825 {
8826 {
8827 \endpgfpicture
8828 }
```

33 The command \SubMatrix

```

8829 \keys_define:nn { nicematrix / sub-matrix }
8830 {
8831   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8832   extra-height .value_required:n = true ,
8833   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8834   left-xshift .value_required:n = true ,
8835   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8836   right-xshift .value_required:n = true ,
8837   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8838   xshift .value_required:n = true ,
8839   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8840   delimiters / color .value_required:n = true ,
8841   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8842   slim .default:n = true ,
8843   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8844   hlines .default:n = all ,
8845   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8846   vlines .default:n = all ,
8847   hvlines .meta:n = { hlines, vlines } ,
8848   hvlines .value_forbidden:n = true
8849 }
8850 \keys_define:nn { nicematrix }
8851 {
8852   SubMatrix .inherit:n = nicematrix / sub-matrix ,
8853   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8854   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8855   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8856 }
```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

8857 \keys_define:nn { nicematrix / SubMatrix }
8858 {
8859   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8860   delimiters / color .value_required:n = true ,
8861   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8862   hlines .default:n = all ,
```

```

8863 vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8864 vlines .default:n = all ,
8865 hvlines .meta:n = { hlines, vlines } ,
8866 hvlines .value_forbidden:n = true ,
8867 name .code:n =
8868   \tl_if_empty:nTF { #1 }
8869     { \@@_error:n { Invalid~name } }
8870     {
8871       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8872       {
8873         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8874           { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
8875           {
8876             \str_set:Nn \l_@@_submatrix_name_str { #1 }
8877             \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8878           }
8879       }
8880       { \@@_error:n { Invalid~name } }
8881     },
8882   name .value_required:n = true ,
8883   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8884   rules .value_required:n = true ,
8885   code .tl_set:N = \l_@@_code_tl ,
8886   code .value_required:n = true ,
8887   unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8888 }

8889 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8890 {
8891   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8892   {
8893     \SubMatrix { #1 } { #2 } { #3 } { #4 }
8894     [
8895       delimiters / color = \l_@@_delimiters_color_tl ,
8896       hlines = \l_@@_submatrix_hlines_clist ,
8897       vlines = \l_@@_submatrix_vlines_clist ,
8898       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8899       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8900       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8901       slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8902       #5
8903     ]
8904   }
8905   \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8906   \ignorespaces
8907 }

8908 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8909   { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8910   { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8911 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8912   {
8913     \seq_gput_right:Ne \g_@@_submatrix_seq
8914     {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8915   { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8916   { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8917   { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8918   { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8919 }
8920 }
```

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).

```

8921 \NewDocumentCommand \@@_compute_i_j:nn
8922   { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8923   { \@@_compute_i_j:nnnn #1 #2 }

8924 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8925 {
8926   \def \l_@@_first_i_tl { #1 }
8927   \def \l_@@_first_j_tl { #2 }
8928   \def \l_@@_last_i_tl { #3 }
8929   \def \l_@@_last_j_tl { #4 }
8930   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8931     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8932   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8933     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8934   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8935     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8936   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8937     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8938 }
```

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`.

```

8939 \hook_gput_code:nnn { begindocument } { . }
8940 {
8941   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m O { } E { _ ^ } { { } { } } }
8942   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
8943     { \@@_sub_matrix:nnnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
8944 }

8945 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8946 {
8947   \group_begin:
```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8948 \@@_compute_i_j:nn { #2 } { #3 }
8949 \int_compare:nNnT { \l_@@_first_i_tl } = { \l_@@_last_i_tl }
8950   { \def \arraystretch { 1 } }
8951 \bool_lazy_or:nnTF
8952   { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
8953   { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
8954   { \@@_error:nn { Construct-too-large } { \SubMatrix } }
8955   {
8956     \str_clear_new:N \l_@@_submatrix_name_str
8957     \keys_set:nn { nicematrix / SubMatrix } { #5 }
8958     \pgfpicture
```

```

8959 \pgfrememberpicturepositiononpage=true
8960 \pgf@relevantforpicturesize=false
8961 \pgfset{inner-sep=\c_zero_dim}
8962 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8963 \dim_set:Nn \l_@@_x_final_dim {-\c_max_dim}

The last value of \int_step_inline:nnn is provided by curryfication.

8964 \bool_if:NTF \l_@@_submatrix_slim_bool
8965   { \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl } }
8966   { \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int } }
8967   {
8968     \cs_if_exist:cT
8969       { \pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8970       {
8971         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8972         \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
8973           { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8974         }
8975       \cs_if_exist:cT
8976         { \pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8977         {
8978           \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8979           \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
8980             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8981           }
8982         }
8983       \dim_compare:nNnTF { \l_@@_x_initial_dim } = { \c_max_dim }
8984         { \@@_error:nn { Impossible~delimiter } { left } }
8985         {
8986           \dim_compare:nNnTF { \l_@@_x_final_dim } = { -\c_max_dim }
8987             { \@@_error:nn { Impossible~delimiter } { right } }
8988             { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8989           }
8990         \endpgfpicture
8991       }
8992     \group_end:
8993     \ignorespaces
8994   }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8995 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8996   {
8997     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8998     \dim_set:Nn \l_@@_y_initial_dim
8999     {
9000       \fp_to_dim:n
9001         {
9002           \pgf@y
9003             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
9004         }
9005     }
9006     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
9007     \dim_set:Nn \l_@@_y_final_dim
9008       { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
9009     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
9010     {
9011       \cs_if_exist:cT
9012         { \pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
9013         {
9014           \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
9015           \dim_set:Nn \l_@@_y_initial_dim
9016             { \dim_max:nn { \l_@@_y_initial_dim } { \pgf@y } }
9017         }

```

```

9018 \cs_if_exist:cT
9019   { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
9020   {
9021     \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
9022     \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
9023       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
9024   }
9025 }
9026 \dim_set:Nn \l_tmpa_dim
9027 {
9028   \l_@@_y_initial_dim - \l_@@_y_final_dim +
9029   \l_@@_submatrix_extra_height_dim - \arrayrulewidth
9030 }
9031 \dim_zero:N \nulldelimeterspace

```

We will draw the rules in the `\SubMatrix`.

```

9032 \group_begin:
9033 \pgfsetlinewidth { 1.1 \arrayrulewidth }
9034 \@@_set_Carc:o \l_@@_rules_color_tl
9035 \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`.

```

9036 \seq_map_inline:Nn \g_@@_cols_vlism_seq
9037 {
9038   \int_compare:nNnT { \l_@@_first_j_tl } < { ##1 }
9039   {
9040     \int_compare:nNnT
9041       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
9042   }

```

First, we extract the value of the abscissa of the rule we have to draw.

```

9043   \@@_qpoint:n { col - ##1 }
9044   \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9045   \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9046   \pgfusepathqstroke
9047 }
9048 }
9049 }

```

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 curryification.

```

9050 \str_if_eq:eeTF { \l_@@_submatrix_vlines_clist } { all }
9051   { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
9052   { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
9053   {
9054     \bool_lazy_and:nnTF
9055       { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
9056       {
9057         \int_compare_p:nNn
9058           { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
9059   {
9060     \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
9061     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9062     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9063     \pgfusepathqstroke
9064   }
9065   { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
9066 }

```

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 curryification.

```

9067 \str_if_eq:eeTF { \l_@@_submatrix_hlines_clist } { all }
9068   { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
9069   { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
9070   {
9071     \bool_lazy_and:nnTF
9072       { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
9073       {
9074         \int_compare_p:nNn
9075           { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 }
9076       {
9077         \qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect \l_tmpa_dim and \l_tmpb_dim .

```
9078 \group_begin:
```

We compute in \l_tmpa_dim the x -value of the left end of the rule.

```

9079   \dim_set:Nn \l_tmpa_dim
9080     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9081   \str_case:nn { #1 }
9082     {
9083       ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9084       [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
9085       \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9086     }
9087     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in \l_tmpb_dim the x -value of the right end of the rule.

```

9088   \dim_set:Nn \l_tmpb_dim
9089     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9090   \str_case:nn { #2 }
9091     {
9092       ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9093       ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
9094       \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9095     }
9096   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9097   \pgfusepathqstroke
9098   \group_end:
9099   { \qerror:nnn { Wrong-line-in-SubMatrix } { horizontal } { ##1 } }
9100 }
9101

```

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).

```

9102 \str_if_empty:NF \l_@@_submatrix_name_str
9103   {
9104     \pgf_rect_node:nnnn \l_@@_submatrix_name_str
9105       \l_@@_x_initial_dim \l_@@_y_initial_dim
9106       \l_@@_x_final_dim \l_@@_y_final_dim
9107   }
9108 \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`.

```

9109 \begin{pgfscope}
9110 \pgftransformshift
9111   {
9112     \pgfpoint
9113       { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9114       { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9115   }
9116 \str_if_empty:NTF \l_@@_submatrix_name_str
9117   { \node_left:nn #1 { } }

```

```

9118 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9119 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9120 \pgftransformshift
9121 {
9122   \pgfpoint
9123     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9124     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9125   }
9126 \str_if_empty:nTF \l_@@_submatrix_name_str
9127   { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9128   {
9129     \@@_node_right:nnnn #2
9130       { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9131   }

```

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`.

```

9132 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9133 \flag_clear_new:N \l_@@_code_flag
9134 \l_@@_code_tl
9135 }

```

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 $i-j$, `row-i`, `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`.

```
9136 \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.

```

9137 \cs_new:Npn \@@_pgfpointanchor:n #1
9138   { \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`.

```

9139 \cs_new:Npn \@@_pgfpointanchor_i:n #1
9140   { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }
9141 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9142   {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```
9143 \str_if_empty:nTF { #1 }
```

First, when the name of the name begins with `\tikz@pp@name`.

```
9144 { \@@_pgfpointanchor_iv:w #2 }
```

And now, when there is no `\tikz@pp@name`.

```

9145 { \@@_pgfpointanchor_ii:n { #1 } }
9146 }
```

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`).

```
9147 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
9148   { \@@_pgfpointanchor_ii:n { #1 } }
```

With the command `\@@_pgfpointanchor_ii:n`, we deal with the actual name of the node (without the `\tikz@pp@name`). First, we have to detect whether it is of the form `i` or 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:nTF` of the package `etl` but, as of now, we do not load `etl`.

```
9149 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }
```

```
9150 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
9151   {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
9152 \str_if_empty:nTF { #2 }
```

First the case where the argument does *not* contain an hyphen.

```
9153   { \@@_pgfpointanchor_iii:n { #1 } }
```

And now the case the argument contains a hyphen. In that case, we have a weird construction because we must retreive the extra hyphen we have added as marker (cf. `\@@_pgfpointanchor_ii:n`).

```
9154   { \@@_pgfpointanchor_iii:w { #1 } #2 }
9155 }
```

The following function is for the case when the name contains an hyphen.

```
9156 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9157   {
```

We have to add the prefix `\@@_env:` “by hand” since we have retreived the potential `\tikz@pp@name`.

```
9158 \@@_env:
9159 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9160 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9161 }
```

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.

```
9162 \tl_const:Nn \c_@@_integers alist_tl
9163 {
9164   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9165   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9166   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9167   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9168 }
```

```
9169 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9170   {
```

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`.

```
9171 \str_case:nVTF { #1 } \c_@@_integers alist_tl
9172 {
9173   \flag_raise:N \l_@@_code_flag
```

We have to add the prefix `\@@_env`: “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9174 \@@_env: -
9175   \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9176     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9177     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9178   }
9179   {
9180     \str_if_eq:eeTF { #1 } { last }
9181     {
9182       \flag_raise:N \l_@@_code_flag
9183       \@@_env: -
9184       \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9185         { \int_eval:n { \l_@@_last_i_tl + 1 } }
9186         { \int_eval:n { \l_@@_last_j_tl + 1 } }
9187       }
9188     { #1 }
9189   }
9190 }
```

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`).

```

9191 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9192 {
9193   \pgfnode
9194     { rectangle }
9195     { east }
9196   {
9197     \nullfont
9198     $ % $
9199     \@@_color:o \l_@@_delimiters_color_tl
9200     \left #1
9201     \vcenter
9202     {
9203       \nullfont
9204       \hrule \height \l_tmpa_dim
9205           \depth \c_zero_dim
9206           \width \c_zero_dim
9207     }
9208     \right .
9209     $ % $
9210   }
9211   { #2 }
9212   { }
9213 }
```

The command `\@@_node_right:nnn` 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.

```

9214 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9215 {
9216   \pgfnode
9217     { rectangle }
9218     { west }
9219   {
9220     \nullfont
9221     $ % $
9222     \colorlet{current-color}{.}
9223     \@@_color:o \l_@@_delimiters_color_tl
9224     \left .
9225     \vcenter
9226     {
```

```

9227     \nullfont
9228     \hrule \@height \l_tmpa_dim
9229         \@depth \c_zero_dim
9230         \@width \c_zero_dim
9231     }
9232     \right #1
9233     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9234     ^ { \color { current-color } \smash { #4 } }
9235     $ % $
9236   }
9237   { #2 }
9238   { }
9239 }

```

34 Les commandes \UnderBrace et \OverBrace

The following commands will be linked to \UnderBrace and \OverBrace in the \CodeAfter.

```

9240 \NewDocumentCommand \@@_UnderBrace { O { } m m m O { } }
9241 {
9242   \@@_brace:nnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9243   \ignorespaces
9244 }

9245 \NewDocumentCommand \@@_OverBrace { O { } m m m O { } }
9246 {
9247   \@@_brace:nnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9248   \ignorespaces
9249 }

9250 \keys_define:nn { nicematrix / Brace }
9251 {
9252   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9253   left-shorten .default:n = true ,
9254   left-shorten .value_forbidden:n = true ,
9255   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9256   right-shorten .default:n = true ,
9257   right-shorten .value_forbidden:n = true ,
9258   shorten .meta:n = { left-shorten , right-shorten } ,
9259   shorten .value_forbidden:n = true ,
9260   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9261   yshift .value_required:n = true ,
9262   yshift .initial:n = \c_zero_dim ,
9263   color .tl_set:N = \l_tmpa_tl ,
9264   color .value_required:n = true ,
9265   unknown .code:n =
9266     \@@_unknown_key:nn
9267       { nicematrix / Brace }
9268       { Unknown~key~for~Brace }
9269 }

```

#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`.

```

9270 \cs_new_protected:Npn \@@_brace:nnnn #1 #2 #3 #4 #5
9271 {
9272   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9273   \@@_compute_i_j:nn { #1 } { #2 }

```

```

9274 \bool_lazy_or:nnTF
9275 { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9276 { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9277 {
9278     \str_if_eq:eeTF { #5 } { under }
9279     { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9280     { \@@_error:nn { Construct-too-large } { \OverBrace } }
9281 }
9282 {
9283     \tl_clear:N \l_tmpa_tl
9284     \keys_set:nn { nicematrix / Brace } { #4 }
9285     \tl_if_empty:N \l_tmpa_tl { \color { \l_tmpa_tl } }
9286     \pgfpicture
9287     \pgfrememberpicturepositiononpagetrue
9288     \pgf@relevantforpicturesizefalse
9289     \bool_if:NT \l_@@_brace_left_shorten_bool
9290     {
9291         \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9292         \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9293         {
9294             \cs_if_exist:cT
9295             { \pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9296             {
9297                 \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9298
9299                 \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9300                 { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9301             }
9302         }
9303     }
9304     \bool_lazy_or:nnT
9305     { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9306     { \dim_compare_p:nNn { \l_@@_x_initial_dim } = { \c_max_dim } }
9307     {
9308         \@@_qpoint:n { col - \l_@@_first_j_tl }
9309         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9310     }
9311     \bool_if:NT \l_@@_brace_right_shorten_bool
9312     {
9313         \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9314         \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9315         {
9316             \cs_if_exist:cT
9317             { \pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9318             {
9319                 \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9320                 \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
9321                 { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9322             }
9323         }
9324     }
9325     \bool_lazy_or:nnT
9326     { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9327     { \dim_compare_p:nNn { \l_@@_x_final_dim } = { - \c_max_dim } }
9328     {
9329         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9330         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9331     }
9332     \pgfset { inner~sep = \c_zero_dim }
9333     \str_if_eq:eeTF { #5 } { under }
9334     { \@@_underbrace_i:n { #3 } }
9335     { \@@_overbrace_i:n { #3 } }
9336 \endpgfpicture

```

```

9337     }
9338   \group_end:
9339 }

The argument is the text to put above the brace.

9340 \cs_new_protected:Npn \@@_overbrace_i:n #1
9341 {
9342   \@@_qpoint:n { row - \l_@@_first_i_tl }
9343   \pgftransformshift
9344   {
9345     \pgfpoint
9346     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9347     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9348   }
9349   \pgfnode
9350   { rectangle }
9351   { south }
9352   {
9353     \vtop
9354     {
9355       \group_begin:
9356       \everycr { }
9357       \halign
9358       {
9359         \hfil ## \hfil \crcr
9360         \bool_if:NTF \l_@@_tabular_bool
9361           { \begin { tabular } { c } #1 \end { tabular } }
9362           { $ \begin { array } { c } #1 \end { array } $ }
9363         \cr
9364         $ \% $
9365         \overbrace
9366         {
9367           \hbox_to_wd:nn
9368             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9369             { }
9370         }
9371         $ \% $
9372         \cr
9373         }
9374       \group_end:
9375     }
9376   }
9377   { }
9378   { }
9379 }

```

The argument is the text to put under the brace.

```

9380 \cs_new_protected:Npn \@@_underbrace_i:n #1
9381 {
9382   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9383   \pgftransformshift
9384   {
9385     \pgfpoint
9386     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9387     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9388   }
9389   \pgfnode
9390   { rectangle }
9391   { north }
9392   {
9393     \group_begin:
9394     \everycr { }
9395     \vbox

```

```

9396   {
9397     \halign
9398     {
9399       \hfil ## \hfil \crcr
9400       $ % $
9401       \underbrace
9402       {
9403         \hbox_to_wd:nn
9404         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9405         { }
9406       }
9407       $ % $
9408       \cr
9409       \bool_if:NTF \l_@@_tabular_bool
9410         { \begin{tabular} { c } #1 \end{tabular} }
9411         { $ \begin{array} { c } #1 \end{array} $ }
9412       \cr
9413     }
9414   }
9415   \group_end:
9416 }
9417 {
9418 }
9419 }
```

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`).

```

9420 \AddToHook { package / tikz / after }
9421 {
9422   \tikzset
9423   {
9424     nicematrix / brace / .style =
9425     {
9426       decoration = { brace , raise = -0.15 em } ,
9427       decorate ,
9428     } ,
```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9429 nicematrix / mirrored-brace / .style =
9430 {
9431   nicematrix / brace ,
9432   decoration = mirror ,
9433 }
9434 }
9435 }
```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9436 \keys_define:nn { nicematrix / Hbrace }
9437 {
9438   color .code:n = ,
9439   horizontal-label .code:n = ,
```

```

9440     horizontal-labels .code:n = ,
9441     shorten .code:n = ,
9442     shorten-start .code:n = ,
9443     shorten-end .code:n = ,
9444     brace-shift .code:n = ,
9445     unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9446 }

```

Here we need an “fully expandable” command.

```

9447 \NewExpandableDocumentCommand { \@@_Hbrace } { O{ } m m }
9448 {
9449   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9450   { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9451   { \@@_error:nn { Hbrace-not-allowed } { \Hbrace } }
9452 }

```

The following command must *not* be protected because of the \Hdotsfor which contains a \multicolumn (whereas the similar command \@@_vbrace:nnn *must* be protected).

```

9453 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9454 {
9455   \int_compare:nNnTF { \c@iRow } < { 2 }
9456   {

```

We recall that \str_if_eq:nnTF is “fully expandable”.

```

9457   \str_if_eq:nnTF { #2 } { * }
9458   {
9459     \bool_set_true:N \l_@@_nullify_dots_bool
9460     \Ldots
9461     [
9462       line-style = nicematrix / brace ,
9463       #1 ,
9464       up =
9465       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9466     ]
9467   }
9468   {
9469     \Hdotsfor
9470     [
9471       line-style = nicematrix / brace ,
9472       #1 ,
9473       up =
9474       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9475     ]
9476     { #2 }
9477   }
9478 }
9479 {
9480   \str_if_eq:nnTF { #2 } { * }
9481   {
9482     \bool_set_true:N \l_@@_nullify_dots_bool
9483     \Ldots
9484     [
9485       line-style = nicematrix / mirrored-brace ,
9486       #1 ,
9487       down =
9488       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9489     ]
9490   }
9491   {
9492     \Hdotsfor
9493     [
9494       line-style = nicematrix / mirrored-brace ,
9495       #1 ,

```

```

9496     down =
9497         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9498     ]
9499     { #2 }
9500   }
9501   ]
9502 \keys_set:nn { nicematrix / Hbrace } { #1 }
9503 }

9504 \NewDocumentCommand { \@@_Vbrace } { O {} m m }
9505 {
9506   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9507   { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9508   { \@@_error:nn { Hbrace-not-allowed } { \Vbrace } }
9509 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

9510 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
9511 {
9512   \int_compare:nNnTF { \c@jCol } < { 2 }
9513   {
9514     \str_if_eq:nnTF { #2 } { * }
9515     {
9516       \bool_set_true:N \l_@@_nullify_dots_bool
9517       \Vdots
9518       [
9519         Vbrace ,
9520         line-style = nicematrix / mirrored-brace ,
9521         #1 ,
9522         down =
9523           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9524       ]
9525     }
9526   {
9527     \Vdotsfor
9528     [
9529       Vbrace ,
9530       line-style = nicematrix / mirrored-brace ,
9531       #1 ,
9532       down =
9533         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9534     ]
9535     { #2 }
9536   }
9537 }
9538 {
9539   \str_if_eq:nnTF { #2 } { * }
9540   {
9541     \bool_set_true:N \l_@@_nullify_dots_bool
9542     \Vdots
9543     [
9544       Vbrace ,
9545       line-style = nicematrix / brace ,
9546       #1 ,
9547       up =
9548         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9549     ]
9550   }
9551   {
9552     \Vdotsfor
9553     [
9554       Vbrace ,

```

```

9555     line-style = nicematrix / brace ,
9556     #1 ,
9557     up =
9558         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9559     ]
9560     { #2 }
9561   }
9562 }
9563 \keys_set:nn { nicematrix / Hbrace } { #1 }
9564 }
```

36 The command TikzEveryCell

```

9565 \bool_new:N \l_@@_not_empty_bool
9566 \bool_new:N \l_@@_empty_bool
9567
9568 \keys_define:nn { nicematrix / TikzEveryCell }
9569 {
9570   not-empty .code:n =
9571     \bool_lazy_or:nnTF
9572     { \l_@@_in_code_after_bool }
9573     { \g_@@_create_cell_nodes_bool }
9574     { \bool_set_true:N \l_@@_not_empty_bool }
9575     { \@@_error:n { detection~of~empty~cells } } ,
9576   not-empty .value_forbidden:n = true ,
9577   empty .code:n =
9578     \bool_lazy_or:nnTF
9579     { \l_@@_in_code_after_bool }
9580     { \g_@@_create_cell_nodes_bool }
9581     { \bool_set_true:N \l_@@_empty_bool }
9582     { \@@_error:n { detection~of~empty~cells } } ,
9583   empty .value_forbidden:n = true ,
9584   unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
9585 }
9586
9587
9588 \NewDocumentCommand { \@@_TikzEveryCell } { O{ } m }
9589 {
9590   \IfPackageLoadedTF { tikz }
9591   {
9592     \group_begin:
9593     \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.

```

9594   \tl_set:Nn \l_tmpa_tl { { #2 } }
9595   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9596     { \@@_for_a_block:nnnnn ##1 }
9597   \@@_all_the_cells:
9598   \group_end:
9599 }
9600 { \@@_error:n { TikzEveryCell~without~tikz } }
9601 }
```

```

9602
9603
9604 \cs_new_protected:Nn \@@_all_the_cells:
9605 {
9606   \int_step_inline:nn \c@iRow
9607   {
9608     \int_step_inline:nn \c@jCol
```

```

9609 {
9610   \cs_if_exist:cF { cell - ##1 - #####1 }
9611   {
9612     \clist_if_in:NeF \l_@@_corners_cells_clist
9613     { ##1 - #####1 }
9614     {
9615       \bool_set_false:N \l_tmpa_bool
9616       \cs_if_exist:cTF
9617         { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
9618         {
9619           \bool_if:NF \l_@@_empty_bool
9620             { \bool_set_true:N \l_tmpa_bool }
9621         }
9622         {
9623           \bool_if:NF \l_@@_not_empty_bool
9624             { \bool_set_true:N \l_tmpa_bool }
9625         }
9626       \bool_if:NT \l_tmpa_bool
9627       {
9628         \@@_block_tikz:onnnn
9629         \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
9630       }
9631     }
9632   }
9633 }
9634 }
9635 }
9636
9637 \cs_new_protected:Nn \@@_for_a_block:nnnn
9638 {
9639   \bool_if:NF \l_@@_empty_bool
9640   {
9641     \@@_block_tikz:onnnn
9642       \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9643   }
9644   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9645 }
9646
9647 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9648 {
9649   \int_step_inline:nnn { #1 } { #3 }
9650   {
9651     \int_step_inline:nnn { #2 } { #4 }
9652       { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9653   }
9654 }

```

37 The command \ShowCellNames

```

9655 \NewDocumentCommand \@@_ShowCellNames { }
9656 {
9657   \bool_if:NT \l_@@_in_code_after_bool
9658   {
9659     \pgfpicture
9660     \pgfrememberpicturepositiononpagetrue
9661     \pgf@relevantforpicturesizefalse
9662     \pgfpathrectanglecorners
9663       { \@@_qpoint:n { 1 } }
9664       {
9665         \@@_qpoint:n
9666           { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
9667       }

```

```

9668 \pgfsetfillcolor { white }
9669 \pgfsetfillcolor { white }
9670 \pgfusepathqfill
9671 \endpgfpicture
9672 }
9673 \dim_gzero_new:N \g_@@_tmpc_dim
9674 \dim_gzero_new:N \g_@@_tmpd_dim
9675 \dim_gzero_new:N \g_@@_tmpe_dim
9676 \int_step_inline:nn { \c@iRow }
9677 {
9678     \bool_if:NTF \l_@@_in_code_after_bool
9679     {
9680         \pgfpicture
9681         \pgfrememberpicturepositiononpagetrue
9682         \pgf@relevantforpicturesizefalse
9683     }
9684     { \begin { pgfpicture } }
9685     \@@_qpoint:n { row - ##1 }
9686     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9687     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9688     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9689     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9690     \bool_if:NTF \l_@@_in_code_after_bool
9691     { \endpgfpicture }
9692     { \end { pgfpicture } }
9693     \int_step_inline:nn { \c@jCol }
9694     {
9695         \hbox_set:Nn \l_tmpa_box
9696         {
9697             \normalfont \Large \sffamily \bfseries
9698             \bool_if:NTF \l_@@_in_code_after_bool
9699                 { \color { red } }
9700                 { \color { red ! 50 } }
9701             ##1 - ####1
9702         }
9703         \bool_if:NTF \l_@@_in_code_after_bool
9704         {
9705             \pgfpicture
9706             \pgfrememberpicturepositiononpagetrue
9707             \pgf@relevantforpicturesizefalse
9708         }
9709         { \begin { pgfpicture } }
9710         \@@_qpoint:n { col - ####1 }
9711         \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9712         \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9713         \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9714         \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9715         \bool_if:NTF \l_@@_in_code_after_bool
9716             { \endpgfpicture }
9717             { \end { pgfpicture } }
9718         \fp_set:Nn \l_tmpa_fp
9719         {
9720             \fp_min:nn
9721             {
9722                 \fp_min:nn
9723                     { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9724                     { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9725                 }
9726                 { 1.0 }
9727             }
9728             \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9729             \pgfpicture
9730             \pgfrememberpicturepositiononpagetrue

```

```

9731     \pgf@relevantforpicturesizefalse
9732     \pgftransformshift
9733     {
9734         \pgfpoint
9735         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpc_dim ) }
9736         { \dim_use:N \g_tmpa_dim }
9737     }
9738     \pgfnode
9739     { rectangle }
9740     { center }
9741     { \box_use:N \l_tmpa_box }
9742     { }
9743     { }
9744     \endpgfpicture
9745 }
9746 }
9747 }

```

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.

```
9748 \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.

```

9749 \bool_new:N \g_@@_footnote_bool
9750 \msg_new:nnnn { nicematrix } { Unknown-key-for-package }
9751 {
9752     You~have~used~the~key~' \l_keys_key_str '~when~loading~nicematrix~
9753     but~that~key~is~unknown. \\
9754     It~will~be~ignored. \\
9755     For~a~list~of~the~available~keys,~type~H~<return>.
9756 }
9757 {
9758     The~available~keys~are~(in~alphabetic~order):~
9759     footnote,~
9760     footnotehyper,~
9761     messages-for-Overleaf,~
9762     renew-dots~and~
9763     renew-matrix.
9764 }
9765 \keys_define:nn { nicematrix }
9766 {
9767     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9768     renew-dots .value_forbidden:n = true ,
9769     renew-matrix .code:n = \@@_renew_matrix: ,
9770     renew-matrix .value_forbidden:n = true ,
9771     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9772     footnote .bool_set:N = \g_@@_footnote_bool ,
9773     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9774     unknown .code:n = \@@_error:n { Unknown-key-for-package }
9775 }
9776 \ProcessKeyOptions

```

```

9777 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9778 {
9779     You~can't~use~the~option~'footnote'~because~the~package~
9780     footnotehyper~has~already~been~loaded.~
9781     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9782     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9783     of~the~package~footnotehyper.\\
9784     The~package~footnote~won't~be~loaded.
9785 }

9786 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9787 {
9788     You~can't~use~the~option~'footnotehyper'~because~the~package~
9789     footnote~has~already~been~loaded.~
9790     If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9791     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9792     of~the~package~footnote.\\
9793     The~package~footnotehyper~won't~be~loaded.
9794 }

```

```

9795 \bool_if:NT \g_@@_footnote_bool
9796 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9797 \IfClassLoadedTF { beamer }
9798 { \bool_set_false:N \g_@@_footnote_bool }
9799 {
9800     \IfPackageLoadedTF { footnotehyper }
9801     { \@@_error:n { footnote~with~footnotehyper~package } }
9802     { \usepackage { footnote } }
9803 }
9804 }

9805 \bool_if:NT \g_@@_footnotehyper_bool
9806 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9807 \IfClassLoadedTF { beamer }
9808 { \bool_set_false:N \g_@@_footnote_bool }
9809 {
9810     \IfPackageLoadedTF { footnote }
9811     { \@@_error:n { footnotehyper~with~footnote~package } }
9812     { \usepackage { footnotehyper } }
9813 }
9814 \bool_set_true:N \g_@@_footnote_bool
9815 }

```

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.

```

9816 \bool_new:N \l_@@_underscore_loaded_bool
9817 \IfPackageLoadedT { underscore }
9818 { \bool_set_true:N \l_@@_underscore_loaded_bool }

```

```

9819 \hook_gput_code:nnn { begindocument } { . }
9820 {
9821   \bool_if:NF \l_@@_underscore_loaded_bool
9822   {
9823     \IfPackageLoadedT { underscore }
9824     { \@@_error:n { underscore~after~nicematrix } }
9825   }
9826 }
```

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 aclist of names of sets of keys and #2 is the error message to send.

```

9827 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
9828 {
9829   \str_set_eq:NN \l_tmpa_str \l_keys_key_str
9830   \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
9831   \str_set:Ne \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
9832   \bool_set_false:N \l_tmpa_bool
9833   \clist_map_inline:nn { #1 }
9834   {
9835     \keys_if_exist:neT { ##1 } { \l_tmpa_str }
9836     {
9837       \@@_error:n { key-with-normal-form-exists }
9838       \bool_set_true:N \l_tmpa_bool
9839       \clist_map_break:
9840     }
9841   }
9842   \bool_if:NF \l_tmpa_bool { \@@_error:n { #2 } }
9843 }
```



```

9844 \@@_msg_new:nn { key-with-normal-form-exists }
9845 {
9846   The~key~'\l_keys_key_str'~does~not~exists.~It~will~be~ignored.\\
9847   Maybe~you~want~to~use~the~key~'\l_tmpa_str'.
9848 }
```



```

9849 \str_const:Ne \c_@@_available_keys_str
9850 {
9851   \bool_if:NT { ! \g_@@_messages_for_Overleaf_bool }
9852   { For~a~list~of~the~available~keys,~type~H~<return>. }
9853 }
```



```

9854 \seq_new:N \g_@@_types_of_matrix_seq
9855 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9856 {
9857   NiceMatrix ,
9858   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9859 }
9860 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9861 { \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:N` 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`.

```

9862 \cs_new_protected:Npn \@@_err_too_many_cols:
9863 {
9864     \seq_if_in:Nf \g_@@_types_of_matrix_seq \g_@@_name_env_str
9865         { \@@_fatal:nn { too-many-cols-for-array } }
9866     \int_compare:nNnT { \l_@@_last_col_int } = { -2 }
9867         { \@@_fatal:n { too-many-cols-for-matrix } }
9868     \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
9869         { \@@_fatal:n { too-many-cols-for-matrix } }
9870     \bool_if:NF \l_@@_last_col_without_value_bool
9871         { \@@_fatal:n { too-many-cols-for-matrix-with-last-col } }
9872 }
```

The following command must *not* be protected since it's used in an error message.

```

9873 \cs_new:Npn \@@_message_hdotsfor:
9874 {
9875     \tl_if_empty:oF \g_@@_Hdotsfor_lines_tl
9876         { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
9877             \token_to_str:N \Hbrace \ is~incorrect. }
9878 }

9879 \cs_new_protected:Npn \@@_Hline_in_cell:
9880     { \@@_fatal:n { Misuse~of~Hline } }

9881 \@@_msg_new:nn { Misuse~of~Hline }
9882 {
9883     Misuse~of~Hline. \\
9884     Error~in~your~row~ \int_eval:n { \c@iRow }. \\
9885     \token_to_str:N \Hline\ must~be~used~only~at~the~beginning~of~a~row.\\
9886     That~error~is~fatal.
9887 }

9888 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9889 {
9890     Incompatible~options.\\
9891     You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~the~same~time.\\
9892     The~output~will~not~be~reliable.
9893 }

9894 \@@_msg_new:nn { Body~alone }
9895 {
9896     \token_to_str:N \Body\ alone. \\
9897     You~have~used~\token_to_str:N \Body\ without~\token_to_str:N \CodeBefore.\\
9898     That~error~is~fatal.
9899 }

9900 \@@_msg_new:nn { key~color~inside }
9901 {
9902     Deleted~key.\\
9903     The~key~'color~inside'~(and~its~alias~'colortbl-like')~has~been~deleted~in
9904     ~'nicematrix'~and~must~not~be~used.\\
9905     This~error~is~fatal.
9906 }

9907 \@@_msg_new:nn { invalid~weight }
9908 {
9909     Unknown~key.\\
9910     The~key~' \l_keys_key_str '~of~your~column~X~is~unknown~and~will~be~ignored.
9911 }

9912 \@@_msg_new:nn { last~col~not~used }
9913 {
9914     Column~not~used.\\
9915     The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
9916     in~your~\@@_full_name_env: .~
9917     However,~you~can~go~on.
9918 }
```

```

9919 \@@_msg_new:nn { too-many-cols-for-matrix-with-last-col }
9920 {
9921   Too-many-columns.\\
9922   In-the-row~ \int_eval:n { \c@iRow },~
9923   you~try~to~use~more~columns~
9924   than~allowed~by~your~ \@@_full_name_env: .
9925   \@@_message_hdotsfor: \\
9926   The~maximal~number~of~columns~is~ \int_eval:n { \l_@@_last_col_int - 1 }~
9927   (plus~the~exterior~columns).~This~error~is~fatal.
9928 }

9929 \@@_msg_new:nn { too-many-cols-for-matrix }
9930 {
9931   Too-many-columns.\\
9932   In-the-row~ \int_eval:n { \c@iRow } ,~
9933   you~try~to~use~more~columns~than~allowed~by~your~ \@@_full_name_env: .
9934   \@@_message_hdotsfor: \\
9935   Recall~that~the~maximal~number~of~columns~for~a~matrix~
9936   (excepted~the~potential~exterior~columns)~is~fixed~by~the~
9937   LaTeX~counter~'MaxMatrixCols'.~
9938   Its~current~value~is~ \int_use:N \c@MaxMatrixCols \\
9939   (use~ \token_to_str:N \setcounter \ to~change~that~value).~
9940   This~error~is~fatal.
9941 }

9942 \@@_msg_new:nn { too-many-cols-for-array }
9943 {
9944   Too-many-columns.\\
9945   In-the-row~ \int_eval:n { \c@iRow } ,~
9946   ~you~try~to~use~more~columns~than~allowed~by~your~
9947   \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal~number~of~columns~is~
9948   \int_use:N \g_@@_static_num_of_col_int \\
9949   \bool_if:nT
9950   { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
9951   { ~(plus~the~exterior~ones) }
9952   since~the~preamble~is~' \g_@@_user_preamble_tl '.\\
9953   This~error~is~fatal.
9954 }

9955 \@@_msg_new:nn { columns-not-used }
9956 {
9957   Columns-not-used.\\
9958   The~preamble~of~your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '.
9959   It~announces~ \int_use:N \g_@@_static_num_of_col_int \\
9960   columns~but~you~only~used~ \int_use:N \c@jCol .\\
9961   The~columns~you~did~not~used~won't~be~created.\\
9962   You~won't~have~similar~warning~till~the~end~of~the~document.
9963 }

9964 \@@_msg_new:nn { empty-preamble }
9965 {
9966   Empty-preamble.\\
9967   The~preamble~of~your~ \@@_full_name_env: \ is~empty.\\
9968   This~error~is~fatal.
9969 }

9970 \@@_msg_new:nn { in-first-col }
9971 {
9972   Erroneous~use.\\
9973   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
9974   That~command~will~be~ignored.
9975 }

9976 \@@_msg_new:nn { in-last-col }
9977 {
9978   Erroneous~use.\\

```

```

9979     You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9980     That~command~will~be~ignored.
9981 }
9982 \@@_msg_new:nn { in-first-row }
9983 {
9984     Erroneous~use.\\
9985     You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9986     That~command~will~be~ignored.
9987 }
9988 \@@_msg_new:nn { in-last-row }
9989 {
9990     Erroneous~use.\\
9991     You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9992     That~command~will~be~ignored.
9993 }
9994 \@@_msg_new:nn { TopRule~without~booktabs }
9995 {
9996     Erroneous~use.\\
9997     You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
9998     That~command~will~be~ignored.
9999 }
10000 \@@_msg_new:nn { TopRule~without~tikz }
10001 {
10002     Erroneous~use.\\
10003     You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
10004     That~command~will~be~ignored.
10005 }
10006 \@@_msg_new:nn { caption~outside~float }
10007 {
10008     Key~caption~forbidden.\\
10009     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
10010     environment~(such~as~\{table\}).~This~key~will~be~ignored.
10011 }
10012 \@@_msg_new:nn { short-caption~without~caption }
10013 {
10014     You~should~not~use~the~key~'short-caption'~without~'caption'.~
10015     However,~your~'short-caption'~will~be~used~as~'caption'.
10016 }
10017 \@@_msg_new:nn { double~closing~delimiter }
10018 {
10019     Double~delimiter.\\
10020     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
10021     delimiter.~This~delimiter~will~be~ignored.
10022 }
10023 \@@_msg_new:nn { delimiter~after~opening }
10024 {
10025     Double~delimiter.\\
10026     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
10027     delimiter.~That~delimiter~will~be~ignored.
10028 }
10029 \@@_msg_new:nn { bad~option~for~line~style }
10030 {
10031     Bad~line~style.\\
10032     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
10033     is~'standard'.~That~key~will~be~ignored.
10034 }
10035 \@@_msg_new:nn { corners~with~no~cell~nodes }
10036 {
10037     Incompatible~keys.\\

```

```

10038 You~can't~use~the~key~'corners'~here~because~the~key~'no-cell-nodes'~
10039 is~in~force.\\
10040 If~you~go~on,~that~key~will~be~ignored.
10041 }

10042 \@@_msg_new:nn { extra-nodes-with-no-cell-nodes }
10043 {
10044   Incompatible~keys.\\
10045   You~can't~create~'extra~nodes'~here~because~the~key~'no-cell-nodes'~
10046 is~in~force.\\
10047   If~you~go~on,~those~extra~nodes~won't~be~created.
10048 }

10049 \@@_msg_new:nn { Identical~notes-in~caption }
10050 {
10051   Identical~tabular~notes.\\
10052   You~can't~put~several~notes~with~the~same~content~in~
10053   \token_to_str:N \caption \ (but~you~can~in~the~main~tabular).\\
10054   If~you~go~on,~the~output~will~probably~be~erroneous.
10055 }

10056 \@@_msg_new:nn { tabularnote-below~the~tabular }
10057 {
10058   \token_to_str:N \tabularnote \ forbidden\\
10059   You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
10060   of~your~tabular~because~the~caption~will~be~composed~below~
10061   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
10062   key~'caption-above'~in~ \token_to_str:N \NiceMatrixOptions .\\
10063   Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
10064   no~similar~error~will~raised~in~this~document.
10065 }

10066 \@@_msg_new:nn { Unknown~key~for~rules }
10067 {
10068   Unknown~key.\\
10069   There~is~only~two~keys~available~here:~width~and~color.\\
10070   Your~key~' \l_keys_key_str ' ~will~be~ignored.
10071 }

10072 \@@_msg_new:nn { Unknown~key~for~Hbrace }
10073 {
10074   Unknown~key.\\
10075   You~have~used~the~key~' \l_keys_key_str ' ~but~the~only~
10076   keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
10077   and~ \token_to_str:N \Vbrace \ are:~'brace-shift',~'color',~
10078   'horizontal-label(s)',~'shorten'~'shorten-end'~
10079   and~'shorten-start'.\\
10080   That~error~is~fatal.
10081 }

10082 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
10083 {
10084   Unknown~key.\\
10085   There~is~only~two~keys~available~here:~
10086   'empty'~and~'not-empty'.\\
10087   Your~key~' \l_keys_key_str ' ~will~be~ignored.
10088 }

10089 \@@_msg_new:nn { Unknown~key~for~rotate }
10090 {
10091   Unknown~key.\\
10092   The~only~key~available~here~is~'c'.\\
10093   Your~key~' \l_keys_key_str ' ~will~be~ignored.
10094 }

10095 \@@_msg_new:nnn { Unknown~key~for~custom-line }
10096 {
10097   Unknown~key.\\

```

```

10098 The~key~' \l_keys_key_str '~is~unknown~in~a~'custom-line'.~
10099 It~you~go~on,~you~will~probably~have~other~errors. \\%
10100 \c_@@_available_keys_str
10101 }
10102 {
10103 The~available~keys~are~(in~alphabetic~order):~
10104 ccommand,~
10105 color,~
10106 command,~
10107 dotted,~
10108 letter,~
10109 multiplicity,~
10110 sep-color,~
10111 tikz,~and~total-width.
10112 }

10113 \@@_msg_new:nnn { Unknown~key~for~xdots }
10114 {
10115 Unknown~key.\\%
10116 The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\%
10117 \c_@@_available_keys_str
10118 }
10119 {
10120 The~available~keys~are~(in~alphabetic~order):~
10121 'color',~
10122 'horizontal(s)-labels',~
10123 'inter',~
10124 'line-style',~
10125 'radius',~
10126 'shorten',~
10127 'shorten-end'~and~'shorten-start'.
10128 }

10129 \@@_msg_new:nn { Unknown~key~for~rowcolors }
10130 {
10131 Unknown~key.\\%
10132 As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
10133 (and~you~try~to~use~' \l_keys_key_str ')\\%
10134 That~key~will~be~ignored.
10135 }

10136 \@@_msg_new:nn { label~without~caption }
10137 {
10138 You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
10139 you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
10140 }

10141 \@@_msg_new:nn { W~warning }
10142 {
10143 Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
10144 (row~ \int_use:N \c@iRow ).~\\%
10145 }

10146 \@@_msg_new:nn { Construct~too~large }
10147 {
10148 Construct~too~large.\\%
10149 Your~command~ \token_to_str:N #1
10150 can't~be~drawn~because~your~matrix~is~too~small.\\%
10151 That~command~will~be~ignored.
10152 }

10153 \@@_msg_new:nn { underscore~after~nicematrix }
10154 {
10155 Problem~with~'underscore'.\\%
10156 The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
10157 You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\%
10158 ' \token_to_str:N \Cdots \token_to_str:N _ '

```

```

10159     \{ n \token_to_str:N \text \{ ~times \} \}'.
10160 }
10161 \@@_msg_new:nn { ampersand~in~light~syntax }
10162 {
10163     Ampersand~forbidden.\\
10164     You~can't~use~an~ampersand~( \token_to_str:N &)~to~separate~columns~because~
10165     ~the~key~'light~syntax'~is~in~force.~This~error~is~fatal.
10166 }
10167 \@@_msg_new:nn { double-backslash~in~light~syntax }
10168 {
10169     Double~backslash~forbidden.\\
10170     You~can't~use~ \token_to_str:N \\
10171     ~to~separate~rows~because~the~key~'light~syntax'~
10172     is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl ' ~
10173     (set~by~the~key~'end~of~row').~This~error~is~fatal.
10174 }
10175 \@@_msg_new:nn { hlines~with~color }
10176 {
10177     Incompatible~keys.\\
10178     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
10179     \token_to_str:N \Block \ when~the~key~'color'~or~'draw'~is~used.\\
10180     However,~you~can~put~several~commands~ \token_to_str:N \Block.\\
10181     Your~key~will~be~discarded.
10182 }
10183 \@@_msg_new:nn { bad~value~for~baseline }
10184 {
10185     Bad~value~for~baseline.\\
10186     The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
10187     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
10188     \int_use:N \g_@@_row_total_int \ or~equal~to~'t',~'c'~or~'b'~or~of~
10189     the~form~'line-i'.\\
10190     A~value~of~1~will~be~used.
10191 }
10192 \@@_msg_new:nn { bad~value~for~baseline-line }
10193 {
10194     Bad~value~for~baseline~with~line.\\
10195     The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
10196     valid.~The~number~of~the~line~must~be~between~1~and~
10197     \int_eval:n { \c@iRow + 1 } \\
10198     A~value~of~'line-1'~will~be~used.
10199 }
10200 \@@_msg_new:nn { detection~of~empty~cells }
10201 {
10202     Problem~with~'not~empty'\\
10203     For~technical~reasons,~you~must~activate~
10204     'create~cell~nodes'~in~ \token_to_str:N \CodeBefore \\
10205     in~order~to~use~the~key~' \l_keys_key_str '.\\
10206     That~key~will~be~ignored.
10207 }
10208 \@@_msg_new:nn { siunitx~not~loaded }
10209 {
10210     siunitx~not~loaded\\
10211     You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
10212     That~error~is~fatal.
10213 }
10214 \@@_msg_new:nn { Invalid~name }
10215 {
10216     Invalid~name.\\
10217     You~can't~give~the~name~' \l_keys_value_tl ' ~to~a~ \token_to_str:N
10218     \SubMatrix \ of~your~ \@@_full_name_env: .\\

```

```

10219 A~name~must~be~accepted~by~the~regular~expression~[A-Za-z] [A-Za-z0-9]*.\\
10220 This~key~will~be~ignored.
10221 }
10222 \@@_msg_new:nn { Hbrace~not~allowed }
10223 {
10224 Command~not~allowed.\\
10225 You~can't~use~the~command~ \token_to_str:N #1
10226 because~you~have~not~loaded~
10227 \IfPackageLoadedTF { tikz }
10228 { the~TikZ~library~'decorations.pathreplacing'.~Use~ }
10229 { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
10230 \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\
10231 That~command~will~be~ignored.
10232 }
10233 \@@_msg_new:nn { Vbrace~not~allowed }
10234 {
10235 Command~not~allowed.\\
10236 You~can't~use~the~command~ \token_to_str:N \Vbrace \\
10237 because~you~have~not~loaded~TikZ~
10238 and~the~TikZ~library~'decorations.pathreplacing'.\\
10239 Use: ~\token_to_str:N \usepackage \{tikz\}~
10240 \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
10241 That~command~will~be~ignored.
10242 }
10243 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
10244 {
10245 Wrong~line.\\
10246 You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
10247 \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
10248 number~is~not~valid.~It~will~be~ignored.
10249 }
10250 \@@_msg_new:nn { Impossible~delimiter }
10251 {
10252 Impossible~delimiter.\\
10253 It's~impossible~to~draw~the~#1~delimiter~of~your~
10254 \token_to_str:N \SubMatrix \ because~all~the~cells~are~empty~
10255 in~that~column.
10256 \bool_if:NT \l_@@_submatrix_slim_bool
10257 { ~Maybe~you~should~try~without~the~key~'slim'. } \\
10258 This~ \token_to_str:N \SubMatrix \ will~be~ignored.
10259 }
10260 \@@_msg_new:nnn { width~without~X~columns }
10261 {
10262 You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
10263 the~preamble~(' \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .\\
10264 That~key~will~be~ignored.
10265 }
10266 {
10267 This~message~is~the~message~'width~without~X~columns'~
10268 of~the~module~'nicematrix'.~
10269 The~experimented~users~can~disable~that~message~with~
10270 \token_to_str:N \msg_redirect_name:nnn .\\
10271 }
10272
10273 \@@_msg_new:nn { key~multiplicity~with~dotted }
10274 {
10275 Incompatible~keys. \\
10276 You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
10277 in~a~'custom-line'.~They~are~incompatible. \\
10278 The~key~'multiplicity'~will~be~discarded.
10279 }

```

```

10280 \@@_msg_new:nn { empty~environment }
10281 {
10282   Empty~environment.\\
10283   Your~ \@@_full_name_env: \ is~empty.~This~error~is~fatal.
10284 }
10285 \@@_msg_new:nn { No~letter~and~no~command }
10286 {
10287   Erroneous~use.\\
10288   Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
10289   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10290   ~'ccommand'~(to~draw~horizontal~rules).\\
10291   However,~you~can~go~on.
10292 }
10293 \@@_msg_new:nn { Forbidden~letter }
10294 {
10295   Forbidden~letter.\\
10296   You~can't~use~the~letter~'#1'~for~a~customized~line.~
10297   It~will~be~ignored.\\
10298   The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10299 }
10300 \@@_msg_new:nn { Several~letters }
10301 {
10302   Wrong~name.\\
10303   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10304   have~used~' \l_@@_letter_str ').\\
10305   It~will~be~ignored.
10306 }
10307 \@@_msg_new:nn { Delimiter~with~small }
10308 {
10309   Delimiter~forbidden.\\
10310   You~can't~put~a~delimiter~in~the~preamble~of~your~
10311   \@@_full_name_env: \
10312   because~the~key~'small'~is~in~force.\\
10313   This~error~is~fatal.
10314 }
10315 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10316 {
10317   Unknown~cell.\\
10318   Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10319   the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10320   can't~be~executed~because~a~cell~doesn't~exist.\\
10321   This~command~ \token_to_str:N \line \ will~be~ignored.
10322 }
10323 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10324 {
10325   Duplicate~name.\\
10326   The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10327   in~this~ \@@_full_name_env: .\\
10328   This~key~will~be~ignored.\\
10329   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10330     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10331 }
10332 {
10333   The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
10334   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10335 }
10336 \@@_msg_new:nn { r~or~l~with~preamble }
10337 {
10338   Erroneous~use.\\
10339   You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
10340   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
```

```

10341     your~ \@@_full_name_env: .\\
10342     This~key~will~be~ignored.
10343 }
10344 \@@_msg_new:nn { Hdotsfor~in~col~0 }
10345 {
10346     Erroneous~use.\\
10347     You~can't~use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\
10348     in~an~exterior~column~of~
10349     the~array.~This~error~is~fatal.
10350 }
10351 \@@_msg_new:nn { bad-corner }
10352 {
10353     Bad~corner.\\
10354     #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10355     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
10356     This~specification~of~corner~will~be~ignored.
10357 }
10358 \@@_msg_new:nn { bad-border }
10359 {
10360     Bad~border.\\
10361     \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
10362     (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block )..~
10363     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10364     also~use~the~key~'tikz'
10365     \IfPackageLoadedF { tikz }
10366     { ~if~you~load~the~LaTeX-package~'tikz' } ).\\
10367     This~specification~of~border~will~be~ignored.
10368 }
10369 \@@_msg_new:nn { TikzEveryCell~without~tikz }
10370 {
10371     TikZ~not~loaded.\\
10372     You~can't~use~ \token_to_str:N \TikzEveryCell \
10373     because~you~have~not~loaded~tikz.~
10374     This~command~will~be~ignored.
10375 }
10376 \@@_msg_new:nn { tikz~key~without~tikz }
10377 {
10378     TikZ~not~loaded.\\
10379     You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N
10380     \Block '~because~you~have~not~loaded~tikz.~
10381     This~key~will~be~ignored.
10382 }
10383 \@@_msg_new:nn { Bad~argument~for~Block }
10384 {
10385     Bad~argument.\\
10386     The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
10387     be~of~the~form~'i-j'~(or~completely~empty)~and~you~have~used:~
10388     '#1'. \\
10389     If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono-cell~(as~if~
10390     the~argument~was~empty).
10391 }
10392 \@@_msg_new:nn { last-col~non~empty~for~NiceArray }
10393 {
10394     Erroneous~use.\\
10395     In~the~ \@@_full_name_env: ,~you~must~use~the~key~
10396     'last-col'~without~value.\\
10397     However,~you~can~go~on~for~this~time~
10398     (the~value~' \l_keys_value_tl '~will~be~ignored).
10399 }
10400 \@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }

```

```

10401 {
10402 Erroneous~use. \\
10403 In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
10404 'last-col'~without~value. \\
10405 However,~you~can~go~on~for~this~time~
10406 (the~value~' \l_keys_value_tl '~will~be~ignored).
10407 }

10408 \@@_msg_new:nn { Block-too-large~1 }
10409 {
10410 Block-too-large. \\
10411 You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
10412 too~small~for~that~block. \\
10413 This~block~and~maybe~others~will~be~ignored.
10414 }

10415 \@@_msg_new:nn { Block-too-large~2 }
10416 {
10417 Block-too-large. \\
10418 The~preamble~of~your~ \@@_full_name_env: \ announces~ \int_use:N
10419 \g_@@_static_num_of_col_int \
10420 columns~but~you~use~only~ \int_use:N \c@jCol \ and~that's~why~a~block~
10421 specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
10422 (&)~at~the~end~of~the~first~row~of~your~ \@@_full_name_env: . \\
10423 This~block~and~maybe~others~will~be~ignored.
10424 }

10425 \@@_msg_new:nn { unknown~column~type }
10426 {
10427 Bad~column~type. \\
10428 The~column~type~'#1'~in~your~ \@@_full_name_env: \
10429 is~unknown. \\
10430 This~error~is~fatal.
10431 }

10432 \@@_msg_new:nn { unknown~column~type~multicolumn }
10433 {
10434 Bad~column~type. \\
10435 The~column~type~'#1'~in~the~command~\token_to_str:N \multicolumn \
10436 ~of~your~ \@@_full_name_env: \
10437 is~unknown. \\
10438 This~error~is~fatal.
10439 }

10440 \@@_msg_new:nn { unknown~column~type~S }
10441 {
10442 Bad~column~type. \\
10443 The~column~type~'S'~in~your~ \@@_full_name_env: \ is~unknown. \\
10444 If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10445 load~that~package. \\
10446 This~error~is~fatal.
10447 }

10448 \@@_msg_new:nn { unknown~column~type~S~multicolumn }
10449 {
10450 Bad~column~type. \\
10451 The~column~type~'S'~in~the~command~\token_to_str:N \multicolumn \
10452 ~of~your~ \@@_full_name_env: \ is~unknown. \\
10453 If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10454 load~that~package. \\
10455 This~error~is~fatal.
10456 }

10457 \@@_msg_new:nn { tabularnote~forbidden }
10458 {
10459 Forbidden~command. \\
10460 You~can't~use~the~command~ \token_to_str:N \tabularnote \
10461 ~here.~This~command~is~available~only~in~

```

```

10462 \{NiceTabular\}, ~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
10463 the~argument~of~a~command~\token_to_str:N \caption \ included~
10464 in~an~environment~\{table\}. \\
10465 This~command~will~be~ignored.
10466 }

10467 \@@_msg_new:nn { borders~forbidden }
10468 {
10469   Forbidden~key.\\
10470   You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \\
10471   because~the~option~'rounded-corners'~
10472   is~in~force~with~a~non~zero~value.\\
10473   This~key~will~be~ignored.
10474 }

10475 \@@_msg_new:nn { bottomrule~without~booktabs }
10476 {
10477   booktabs~not~loaded.\\
10478   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
10479   loaded~'booktabs'.\\
10480   This~key~will~be~ignored.
10481 }

10482 \@@_msg_new:nn { enumitem~not~loaded }
10483 {
10484   enumitem~not~loaded. \\
10485   You~can't~use~the~command~ \token_to_str:N \tabularnote \\
10486   ~because~you~haven't~loaded~'enumitem'. \\
10487   All~the~commands~ \token_to_str:N \tabularnote \ will~be~
10488   ignored~in~the~document.
10489 }

10490 \@@_msg_new:nn { tikz~without~tikz }
10491 {
10492   Tikz~not~loaded. \\
10493   You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
10494   loaded.~If~you~go~on,~that~key~will~be~ignored.
10495 }

10496 \@@_msg_new:nn { tikz~in~custom-line~without~tikz }
10497 {
10498   Tikz~not~loaded. \\
10499   You~have~used~the~key~'tikz'~in~the~definition~of~a~
10500   customized-line~(with~'custom-line')~but~tikz~is~not~loaded.~
10501   You~can~go~on~but~you~will~have~another~error~if~you~actually~
10502   use~that~custom-line.
10503 }

10504 \@@_msg_new:nn { tikz~in~borders~without~tikz }
10505 {
10506   Tikz~not~loaded. \\
10507   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
10508   command~' \token_to_str:N \Block ')~but~tikz~is~not~loaded.~
10509   That~key~will~be~ignored.
10510 }

10511 \@@_msg_new:nn { color~in~custom-line~with~tikz }
10512 {
10513   Erroneous~use.\\
10514   In~a~'custom-line',~you~have~used~both~'tikz'~and~'color',~
10515   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
10516   The~key~'color'~will~be~discarded.
10517 }

10518 \@@_msg_new:nn { Wrong~last~row }
10519 {
10520   Wrong~number.\\
10521   You~have~used~'last-row= \int_use:N \l_@@_last_row_int '~-but~your~

```

```

10522 \@@_full_name_env: \ seems-to-have~ \int_use:N \c@iRow \ rows.~
10523 If~you-go-on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
10524 last~row~but~you~should~correct~your~code.~You~can~avoid~this~
10525 problem~by~using~'last-row'~without~value~(more~compilations~
10526 might~be~necessary).
10527 }

10528 \@@_msg_new:nn { Yet-in-env }
10529 {
10530   Nested-environments.\\
10531   Environments-of-nicematrix-can't-be-nested.\\
10532   This-error-is-fatal.
10533 }

10534 \@@_msg_new:nn { Outside~math~mode }
10535 {
10536   Outside~math~mode.\\
10537   The~\@@_full_name_env: \ can-be-used-only-in~math~mode~
10538   (and-not-in~ \token_to_str:N \vcenter ).\\
10539   This~error~is~fatal.
10540 }

10541 \@@_msg_new:nn { One-letter~allowed }
10542 {
10543   Bad~name.\\
10544   The~value~of~key~' \l_keys_key_str ' ~must~be~of~length~1~and~
10545   you~have~used~' \l_keys_value_tl '.\\
10546   It~will~be~ignored.
10547 }

10548 \@@_msg_new:nn { TabularNote~in~CodeAfter }
10549 {
10550   Environment~\{TabularNote\}~forbidden.\\
10551   You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
10552   but~*before*~the~ \token_to_str:N \CodeAfter . \\
10553   This~environment~\{TabularNote\}~will~be~ignored.
10554 }

10555 \@@_msg_new:nn { varwidth~not~loaded }
10556 {
10557   varwidth~not~loaded.\\
10558   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
10559   loaded.\\
10560   Your~column~will~behave~like~'p'.
10561 }

10562 \@@_msg_new:nn { varwidth~not~loaded~in~X }
10563 {
10564   varwidth~not~loaded.\\
10565   You~can't~use~the~key~'V'~in~your~column~'X'~
10566   because~'varwidth'~is~not~loaded.\\
10567   It~will~be~ignored. \\
10568 }

10569 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
10570 {
10571   Unknown~key.\\
10572   Your~key~' \l_keys_key_str ' ~is~unknown~for~a~rule.\\
10573   \c_@@_available_keys_str
10574 }
10575 {
10576   The~available~keys~are~(in~alphabetic~order):~
10577   color,~
10578   dotted,~
10579   multiplicity,~
10580   sep-color,~
10581   tikz,~and~total-width.
10582 }

```

```

10583
10584 \@@_msg_new:nnn { Unknown~key~for~Block }
10585 {
10586     Unknown~key. \\
10587     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10588     \token_to_str:N \Block . \\
10589     It~will~be~ignored. \\
10590     \c_@@_available_keys_str
10591 }
10592 {
10593     The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10594     b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
10595     opacity,~rounded~corners,~r,~respect~arraystretch,~t,~T,~tikz,~transparent~
10596     and~vlines.
10597 }
10598 \@@_msg_new:nnn { Unknown~key~for~Brace }
10599 {
10600     Unknown~key.\\
10601     The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
10602     \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
10603     It~will~be~ignored. \\
10604     \c_@@_available_keys_str
10605 }
10606 {
10607     The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
10608     right~shorten,~shorten~(which~fixes~both~left~shorten~and~
10609     right~shorten)~and~yshift.
10610 }
10611 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10612 {
10613     Unknown~key.\\
10614     The~key~' \l_keys_key_str '~is~unknown.\\
10615     It~will~be~ignored. \\
10616     \c_@@_available_keys_str
10617 }
10618 {
10619     The~available~keys~are~(in~alphabetic~order):~
10620     delimiters/color,~
10621     rules~(with~the~subkeys~'color'~and~'width'),~
10622     sub-matrix~(several~subkeys)~
10623     and~xdots~(several~subkeys).~
10624     The~latter~is~for~the~command~ \token_to_str:N \line .
10625 }
10626 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10627 {
10628     Unknown~key.\\
10629     The~key~' \l_keys_key_str '~is~unknown.\\
10630     It~will~be~ignored. \\
10631     \c_@@_available_keys_str
10632 }
10633 {
10634     The~available~keys~are~(in~alphabetic~order):~
10635     create-cell-nodes,~
10636     delimiters/color~and~
10637     sub-matrix~(several~subkeys).
10638 }
10639 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10640 {
10641     Unknown~key.\\
10642     The~key~' \l_keys_key_str '~is~unknown.\\
10643     That~key~will~be~ignored. \\
10644     \c_@@_available_keys_str

```

```

10645 }
10646 {
10647 The~available~keys~are~(in~alphabetic~order):~
10648 'delimiters/color',~
10649 'extra-height',~
10650 'hlines',~
10651 'hvlines',~
10652 'left-xshift',~
10653 'name',~
10654 'right-xshift',~
10655 'rules'~(with~the~subkeys~'color'~and~'width'),~
10656 'slim',~
10657 'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10658 and~'right-xshift').\\
10659 }
10660 \\@@_msg_new:nnn { Unknown~key~for~notes }
10661 {
10662 Unknown~key.\\
10663 The~key~' \l_keys_key_str '~is~unknown.\\\
10664 That~key~will~be~ignored. \\\
10665 \c_@@_available_keys_str
10666 }
10667 {
10668 The~available~keys~are~(in~alphabetic~order):~
10669 bottomrule,~
10670 code-after,~
10671 code-before,~
10672 detect-duplicates,~
10673 enumitem-keys,~
10674 enumitem-keys-para,~
10675 para,~
10676 label-in-list,~
10677 label-in-tabular~and~
10678 style.
10679 }
10680 \\@@_msg_new:nnn { Unknown~key~for~RowStyle }
10681 {
10682 Unknown~key.\\
10683 The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10684 \token_to_str:N \RowStyle . \\\
10685 That~key~will~be~ignored. \\\
10686 \c_@@_available_keys_str
10687 }
10688 {
10689 The~available~keys~are~(in~alphabetic~order):~
10690 bold,~
10691 cell-space-top-limit,~
10692 cell-space-bottom-limit,~
10693 cell-space-limits,~
10694 color,~
10695 fill~(alias:~rowcolor),~
10696 nb-rows,~
10697 opacity~and~
10698 rounded-corners.
10699 }
10700 \\@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10701 {
10702 Unknown~key.\\
10703 The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10704 \token_to_str:N \NiceMatrixOptions . \\\
10705 That~key~will~be~ignored. \\\
10706 \c_@@_available_keys_str
10707 }

```

```

10708 {
10709   The~available~keys~are~(in~alphabetic~order):~
10710   &-in-blocks,~
10711   allow-duplicate-names,~
10712   ampersand-in-blocks,~
10713   caption-above,~
10714   cell-space-bottom-limit,~
10715   cell-space-limits,~
10716   cell-space-top-limit,~
10717   code-for-first-col,~
10718   code-for-first-row,~
10719   code-for-last-col,~
10720   code-for-last-row,~
10721   corners,~
10722   custom-key,~
10723   create-extra-nodes,~
10724   create-medium-nodes,~
10725   create-large-nodes,~
10726   custom-line,~
10727   delimiters~(several~subkeys),~
10728   end-of-row,~
10729   first-col,~
10730   first-row,~
10731   hlines,~
10732   hvlines,~
10733   hvlines-except-borders,~
10734   last-col,~
10735   last-row,~
10736   left-margin,~
10737   light-syntax,~
10738   light-syntax-expanded,~
10739   matrix/columns-type,~
10740   no-cell-nodes,~
10741   notes~(several~subkeys),~
10742   nullify-dots,~
10743   pgf-node-code,~
10744   renew-dots,~
10745   renew-matrix,~
10746   respect-arraystretch,~
10747   rounded-corners,~
10748   right-margin,~
10749   rules~(with~the~subkeys~'color'~and~'width'),~
10750   small,~
10751   sub-matrix~(several~subkeys),~
10752   vlines,~
10753   xdots~(several~subkeys).
10754 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no `l` and `r`.

```

10755 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10756 {
10757   Unknown~key.\\
10758   The~key~' \l_keys_key_str ' ~is~unknown~for~the~environment~\\
10759   \{NiceArray\}. \\
10760   That~key~will~be~ignored. \\
10761   \c_@@_available_keys_str
10762 }
10763 {
10764   The~available~keys~are~(in~alphabetic~order):~
10765   &-in-blocks,~
10766   ampersand-in-blocks,~
10767   b,~
10768   baseline,~

```

```

10769   c,~
10770   cell-space-bottom-limit,~
10771   cell-space-limits,~
10772   cell-space-top-limit,~
10773   code-after,~
10774   code-for-first-col,~
10775   code-for-first-row,~
10776   code-for-last-col,~
10777   code-for-last-row,~
10778   columns-width,~
10779   corners,~
10780   create-extra-nodes,~
10781   create-medium-nodes,~
10782   create-large-nodes,~
10783   extra-left-margin,~
10784   extra-right-margin,~
10785   first-col,~
10786   first-row,~
10787   hlines,~
10788   hvlines,~
10789   hvlines-except-borders,~
10790   last-col,~
10791   last-row,~
10792   left-margin,~
10793   light-syntax,~
10794   light-syntax-expanded,~
10795   name,~
10796   no-cell-nodes,~
10797   nullify-dots,~
10798   pgf-node-code,~
10799   renew-dots,~
10800   respect-arraystretch,~
10801   right-margin,~
10802   rounded-corners,~
10803   rules~(with~the~subkeys~'color'~and~'width'),~
10804   small,~
10805   t,~
10806   vlines,~
10807   xdots/color,~
10808   xdots/shorten-start,~
10809   xdots/shorten-end,~
10810   xdots/shorten-and~
10811   xdots/line-style.
10812 }

```

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`).

```

10813 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10814 {
10815   Unknown~key.\\
10816   The~key~' \l_keys_key_str ' ~is~unknown~for~the~
10817   \@@_full_name_env: . \\%
10818   That~key~will~be~ignored. \\%
10819   \c_@@_available_keys_str
10820 }
10821 {
10822   The~available~keys~are~(in~alphabetic~order):~%
10823   &-in-blocks,~
10824   ampersand-in-blocks,~
10825   b,~
10826   baseline,~
10827   c,~
10828   cell-space-bottom-limit,~

```

```

10829   cell-space-limits,~
10830   cell-space-top-limit,~
10831   code-after,~
10832   code-for-first-col,~
10833   code-for-first-row,~
10834   code-for-last-col,~
10835   code-for-last-row,~
10836   columns-type,~
10837   columns-width,~
10838   corners,~
10839   create-extra-nodes,~
10840   create-medium-nodes,~
10841   create-large-nodes,~
10842   extra-left-margin,~
10843   extra-right-margin,~
10844   first-col,~
10845   first-row,~
10846   hlines,~
10847   hvlines,~
10848   hvlines-except-borders,~
10849   l,~
10850   last-col,~
10851   last-row,~
10852   left-margin,~
10853   light-syntax,~
10854   light-syntax-expanded,~
10855   name,~
10856   no-cell-nodes,~
10857   nullify-dots,~
10858   pgf-node-code,~
10859   r,~
10860   renew-dots,~
10861   respect-arraystretch,~
10862   right-margin,~
10863   rounded-corners,~
10864   rules~(with~the~subkeys~'color'~and~'width'),~
10865   small,~
10866   t,~
10867   vlines,~
10868   xdots/color,~
10869   xdots/shorten-start,~
10870   xdots/shorten-end,~
10871   xdots/shorten-and~
10872   xdots/line-style.
10873 }

10874 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10875 {
10876   Unknown~key.\\
10877   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~\\
10878   \{NiceTabular\}. \\
10879   That~key~will~be~ignored. \\
10880   \c_@@_available_keys_str
10881 }
10882 {
10883   The~available~keys~are~(in~alphabetic~order):~\\
10884   &-in-blocks,~
10885   ampersand-in-blocks,~
10886   b,~
10887   baseline,~
10888   c,~
10889   caption,~
10890   cell-space-bottom-limit,~
10891   cell-space-limits,~

```

```

10892   cell-space-top-limit,~
10893   code-after,~
10894   code-for-first-col,~
10895   code-for-first-row,~
10896   code-for-last-col,~
10897   code-for-last-row,~
10898   columns-width,~
10899   corners,~
10900   custom-line,~
10901   create-extra-nodes,~
10902   create-medium-nodes,~
10903   create-large-nodes,~
10904   extra-left-margin,~
10905   extra-right-margin,~
10906   first-col,~
10907   first-row,~
10908   hlines,~
10909   hvlines,~
10910   hvlines-except-borders,~
10911   label,~
10912   last-col,~
10913   last-row,~
10914   left-margin,~
10915   light-syntax,~
10916   light-syntax-expanded,~
10917   name,~
10918   no-cell-nodes,~
10919   notes-(several~subkeys),~
10920   nullify-dots,~
10921   pgf-node-code,~
10922   renew-dots,~
10923   respect-arraystretch,~
10924   right-margin,~
10925   rounded-corners,~
10926   rules~(with~the~subkeys~'color'~and~'width'),~
10927   short-caption,~
10928   t,~
10929   tabularnote,~
10930   vlines,~
10931   xdots/color,~
10932   xdots/shorten-start,~
10933   xdots/shorten-end,~
10934   xdots/shorten-and~
10935   xdots/line-style.
10936 }

10937 \@@_msg_new:nnn { Duplicate~name }
10938 {
10939   Duplicate~name.\\
10940   The~name~' \l_keys_value_tl ' ~is~already~used~and~you~shouldn't~use~
10941   the~same~environment~name~twice.~You~can~go~on,~but,~
10942   maybe,~you~will~have~incorrect~results~especially~
10943   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10944   message~again,~use~the~key~'allow-duplicate-names'~in~
10945   ' \token_to_str:N \NiceMatrixOptions '.\\
10946   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10947   { For~a~list~of~the~names~already~used,~type~H~<return>. }
10948 }
10949 {
10950   The~names~already~defined~in~this~document~are:~
10951   \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
10952 }

10953 \@@_msg_new:nn { caption-above-in-env }
10954 {

```

```

10955     The~key~'caption-above'~must~be~used~in~\token_to_str:N \NiceMatrixOptions.\\
10956     That~key~will~be~ignored.
10957 }
10958 \@@_msg_new:nn { show-cell-names }
10959 {
10960     There~is~no~key~'show-cell-names'~in~nicematrix.\\
10961     You~should~use~the~command~\token_to_str:N \ShowCellNames\\
10962     in~the~\token_to_str:N \CodeBefore~or~the~\token_to_str:N
10963     \CodeAfter. \\
10964     That~key~will~be~ignored.
10965 }
10966 \@@_msg_new:nn { Option~auto~for~columns~width }
10967 {
10968     Erroneous~use.\\
10969     You~can't~give~the~value~'auto'~to~the~key~'columns~width'~here.~
10970     That~key~will~be~ignored.
10971 }
10972 \@@_msg_new:nn { NiceTabularX~without~X }
10973 {
10974     NiceTabularX~without~X.\\
10975     You~should~not~use~\{NiceTabularX\}~without~X~columns.\\
10976     However,~you~can~go~on.
10977 }
10978 \@@_msg_new:nn { Preamble~forgotten }
10979 {
10980     Preamble~forgotten.\\
10981     You~have~probably~forgotten~the~preamble~of~your~
10982     \@@_full_name_env: . \\
10983     This~error~is~fatal.
10984 }
10985 \@@_msg_new:nn { Invalid~col~number }
10986 {
10987     Invalid~column~number.\\
10988     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \\
10989     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.
10990 }
10991 \@@_msg_new:nn { Invalid~row~number }
10992 {
10993     Invalid~row~number.\\
10994     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \\
10995     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.
10996 }
10997 \@@_define_com:NNN p ( )
10998 \@@_define_com:NNN b [ ]
10999 \@@_define_com:NNN v | |
11000 \@@_define_com:NNN V \| \|
11001 \@@_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 \tabularnote	20
6	Command for creation of rectangle nodes	25
7	The options	26
8	Important code used by {NiceArrayWithDelims}	37
9	The \CodeBefore	51
10	The environment {NiceArrayWithDelims}	55
11	Construction of the preamble of the array	60
12	The redefinition of \multicolumn	77
13	The environment {NiceMatrix} and its variants	94
14	{NiceTabular}, {NiceTabularX} and {NiceTabular*}	95
15	After the construction of the array	97
16	We draw the dotted lines	103
17	The actual instructions for drawing the dotted lines with Tikz	118
18	User commands available in the new environments	124
19	The command \line accessible in code-after	130
20	The command \RowStyle	132
21	Colors of cells, rows and columns	135
22	The vertical and horizontal rules	147
23	The empty corners	164
24	The environment {NiceMatrixBlock}	166
25	The extra nodes	167
26	The blocks	172
27	How to draw the dotted lines transparently	198
28	Automatic arrays	199
29	The redefinition of the command \dotfill	200
30	The command \diagbox	200

31	The keyword \CodeAfter	202
32	The delimiters in the preamble	203
33	The command \SubMatrix	204
34	Les commandes \UnderBrace et \OverBrace	213
35	The commands HBrace et VBrace	216
36	The command TikzEveryCell	219
37	The command \ShowCellNames	220
38	We process the options at package loading	222
39	About the package underscore	223
40	Error messages of the package	224