

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

March 11, 2026

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

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

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

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

*This document corresponds to the version 7.8 of `nicematrix`, at the date of 2026/03/11.

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

```

20 \RequirePackage { amsmath }

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

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

```

We test whether the current class is `revtex4-1` (deprecated) or `revtex4-2` because the version 4.2f of `revtex4-2` is incompatible with `nicematrix`.

```

58 \IfClassLoadedTF { revtex4-1 }
59 { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
60 {
61   \IfClassLoadedTF { revtex4-2 }
62   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
63   {

```

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

```

64     \cs_if_exist:NT \rvtx@ifformat@geq
65     { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
66     { \bool_const:Nn \c_@@_revtex_bool { \c_false_bool } }
67   }
68 }
69 \@@_msg_new:nn { class~revtex }
70 {
71   You~can't~use~this~version~of~'nicematrix'~in~a~class~of~REVTeX~because~
72   REVTeX~is~*not*~compatible~with~recent~versions~of~LaTeX.~Sorry...\\
73   The~package~'nicematrix'~won't~be~loaded.
74 }
75 \bool_if:NT \c_@@_revtex_bool
76 { \msg_critical:nn { nicematrix } { class~revtex } }

```

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`).

```

77 \cs_new_protected:Npn \@@_collect_options:n #1
78 {
79   \peek_meaning:NTF [
80     { \@@_collect_options:nw { #1 } }
81     { #1 { } }
82   }

```

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

```

83 \NewDocumentCommand \@@_collect_options:nw { m r[] }
84 { \@@_collect_options:nn { #1 } { #2 } }
85
86 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
87 {
88   \peek_meaning:NTF [
89     { \@@_collect_options:nnw { #1 } { #2 } }
90     { #1 { #2 } }
91   }
92
93 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
94 { \@@_collect_options:nn { #1 } { #2 , #3 } }

```

3 Technical definitions

Here are definitions that have been added to the LaTeX kernel in February 2006.
The following constants are defined only for efficiency in the tests.

```

95 \tl_const:Nn \c_@@_c_tl { c }
96 \tl_const:Nn \c_@@_l_tl { l }
97 \tl_const:Nn \c_@@_r_tl { r }
98 \tl_const:Nn \c_@@_all_tl { all }
99 \tl_const:Nn \c_@@_dot_tl { . }
100 \str_const:Nn \c_@@_r_str { r }
101 \str_const:Nn \c_@@_c_str { c }
102 \str_const:Nn \c_@@_l_str { l }

103 \tl_const:Nn \c_@@_brace_tl { nicematrix/brace }
104 \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).

```

105 \tl_new:N \l_@@_argspec_tl

106 \cs_generate_variant:Nn \seq_set_split:Nnn { N o }
107 \cs_generate_variant:Nn \str_set:Nn { N o }
108 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
109 \prg_generate_conditional_variant:Nnn \clist_if_in:Nn { N e } { T , F, TF }
110 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { e } { T }
111 \prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
112 \cs_generate_variant:Nn \dim_min:nn { v }
113 \cs_generate_variant:Nn \dim_max:nn { v }

114 \AtBeginDocument
115 {
116   \IfPackageLoadedTF { tikz }
117   {

```

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 `\AtBeginDocument` the correct version of some commands. The tokens `\exp_not:N` are mandatory.

```

118   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
119   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
120 }
121 {
122   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
123   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
124 }
125 }

```

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.

```

126 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
127 {
128   \iow_now:Nn \@mainaux
129   {

```

```

130     \ExplSyntaxOn
131     \cs_if_free:NT \pgfsyspdfmark
132     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
133     \ExplSyntaxOff
134   }
135   \cs_gset_eq:NN \@_provide_pgfsyspdfmark: \prg_do_nothing:
136 }

```

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

```

137 \ProvideDocumentCommand \iddots { }
138 {
139   \mathinner
140   {
141     \mkern 1 mu
142     \box_move_up:nn { 1 pt } { \hbox { . } }
143     \mkern 2 mu
144     \box_move_up:nn { 4 pt } { \hbox { . } }
145     \mkern 2 mu
146     \box_move_up:nn { 7 pt }
147     { \vbox:n { \kern 7 pt \hbox { . } } }
148     \mkern 1 mu
149   }
150 }

```

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.

```

151 \AtBeginDocument
152 {
153   \IfPackageLoadedT { booktabs }
154   {
155     \iow_now:Nn \@mainaux
156     {
157       \ExplSyntaxOn
158       \cs_if_exist_use:NT \nicematrix@redefine@check@rerun
159       \ExplSyntaxOff
160     }
161   }
162 }
163 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
164 {
165   \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`).

```

166   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
167   {
168     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } { 1 } { 3 } }
169     { \@_old_pgfutil@check@rerun { ##1 } { ##2 } }
170   }
171 }

```

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

```

172 \AtBeginDocument

```

```

173 {
174   \cs_set_protected:Npe \@@_everycr:
175   {
176     \IfPackageLoadedTF { colortbl } \CT@everycr \everycr
177     { \noalign { \@@_in_everycr: } }
178   }
179   \IfPackageLoadedTF { colortbl }
180   {
181     \cs_new_eq:NN \@@_old_cellcolor: \cellcolor
182     \cs_new_eq:NN \@@_old_rowcolor: \rowcolor
183     \cs_new_protected:Npn \@@_revert_colortbl:
184     {
185       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
186       {
187         \cs_set_eq:NN \cellcolor \@@_old_cellcolor:
188         \cs_set_eq:NN \rowcolor \@@_old_rowcolor:
189       }
190     }

```

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).

```

191     \cs_new_protected:Npn \@@_replace_columncolor:
192     {
193       \tl_replace_all:Nnn \g_@@_array_preamble_tl
194       \columncolor
195       \@@_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.

```

196     }
197   }
198   {
199     \cs_new_protected:Npn \@@_revert_colortbl: { }
200     \cs_new_protected:Npn \@@_replace_columncolor:
201     { \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.

```

202     \def \CT@arc@ { }
203     \def \arrayrulecolor #1 # { \CT@arc { #1 } }
204     \def \CT@arc #1 #2
205     {
206       \dim_compare:nNnT \baselineskip = \c_zero_dim { \noalign }
207       { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
208     }

```

Idem for `\CT@drs@`.

```

209     \def \doublerulesepcolor #1 # { \CT@drs { #1 } }
210     \def \CT@drs #1 #2
211     {
212       \dim_compare:nNnT \baselineskip = \c_zero_dim { \noalign }
213       { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
214     }
215     \def \hline
216     {
217       \noalign { \ifnum 0 = ` } \fi
218       \cs_set_eq:NN \hskip \vskip
219       \cs_set_eq:NN \vrule \hrule
220       \cs_set_eq:NN \@width \@height
221       { \CT@arc@ \vline }
222       \futurelet \reserved@a
223       \@xhline
224     }

```

```

225     }
226 }

```

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.

```

227 \cs_set_nopar:Npn \@@_standard_cline: #1 { \@@_standard_cline:w #1 \q_stop }
228 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
229 {
230   \int_if_zero:nT \l_@@_first_col_int { \omit & }
231   \int_compare:nNnT { #1 } > 1
232     { \multispan { \int_eval:n { #1 - 1 } } & }
233   \multispan { \int_eval:n { #2 - #1 + 1 } }
234   {
235     \CT@arc@
236     \leaders \hrule \@height \arrayrulewidth \hfill

```

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

```

237     \skip_horizontal:N \c_zero_dim
238 }

```

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`.

```

239 \everycr { }
240 \cr
241 \noalign { \skip_vertical:n { - \arrayrulewidth } }
242 }

```

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.

```

243 \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`.

```

244 { \@@_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*.

```

245 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
246 \cs_generate_variant:Nn \@@_cline_i:nn { e }
247 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
248 {
249   \tl_if_empty:nTF { #3 }
250     { \@@_cline_iii:w #1|#2-#2 \q_stop }
251     { \@@_cline_ii:w #1|#2-#3 \q_stop }
252 }
253 \cs_set:Npn \@@_cline_ii:w #1|#2-#3- \q_stop
254 { \@@_cline_iii:w #1|#2-#3 \q_stop }
255 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
256 {

```

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).

```

257   \int_compare:nNnT { #1 } < { #2 }
258     { \multispan { \int_eval:n { #2 - #1 } } & }
259   \multispan { \int_eval:n { #3 - #2 + 1 } }
260   {
261     \CT@arc@

```

¹See question 99041 on TeX StackExchange.

```

262     \leaders \hrule \@height \arrayrulewidth \hfill
263     \skip_horizontal:N \c_zero_dim
264 }

```

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

```

265 \peek_meaning_remove_ignore_spaces:NTF \cline
266 { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
267 { \everycr { } \cr }
268 }

```

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

```

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

270 \cs_new_protected:Npn \@@_set_CTarc:n #1
271 {
272   \tl_if_blank:nF { #1 }
273   {
274     \tl_if_head_eq_meaning:nNTF { #1 } [
275       { \def \CT@arc@ { \color #1 } }
276       { \def \CT@arc@ { \color { #1 } } }
277     ]
278   }
279 \cs_generate_variant:Nn \@@_set_CTarc:n { o }

```

```

280 \cs_new_protected:Npn \@@_set_CTdrsc:n #1
281 {
282   \tl_if_head_eq_meaning:nNTF { #1 } [
283     { \def \CT@drsc@ { \color #1 } }
284     { \def \CT@drsc@ { \color { #1 } } }
285   ]

```

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

```

286 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
287 {
288   \tl_if_head_eq_meaning:nNTF { #2 } [
289     { #1 #2 }
290     { #1 { #2 } }
291   ]
292 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```

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

```

293 \cs_new_protected:Npn \@@_color:n #1
294 { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }
295 \cs_generate_variant:Nn \@@_color:n { o }

```

```

296 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
297 {
298   \tl_set_rescan:Nno
299   #1
300   {
301     \char_set_catcode_other:N >
302     \char_set_catcode_other:N <
303   }
304   #1
305 }

```


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

```

306 \dim_new:N \l_@@_tmpc_dim
307 \dim_new:N \l_@@_tmpd_dim

308 \tl_new:N \l_@@_tmpc_tl
309 \tl_new:N \l_@@_tmpd_tl

310 \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.

```

311 \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).

```

312 \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.

```

313 \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.

```

314 \box_new:N \l_@@_the_array_box

```

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

```

315 \cs_new_protected:Npn \@@_qpoint:n #1
316 { \pgfpointanchor { \@@_env: - #1 } { center } }

```

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

```

317 \bool_new:N \l_@@_tabular_bool

```

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

```

318 \bool_new:N \g_@@_delims_bool
319 \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.

```

320 \bool_new:N \l_@@_preamble_bool
321 \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.

```

322 \bool_new:N \l_@@_NiceMatrix_without_vlines_bool

```

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

```
323 \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.

```
324 \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).

```
325 \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.).

```
326 \dim_new:N \l_@@_col_width_dim
327 \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.

```
328 \int_new:N \g_@@_row_total_int
329 \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).

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

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

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

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

```
332 \tl_new:N \l_@@_hpos_cell_tl
333 \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`.

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

Idem for the mono-row blocks.

```
335 \dim_new:N \g_@@_blocks_ht_dim
336 \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}`).

```
337 \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`.

```
338 \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.

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

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

```
340 \bool_new:N \l_@@_notes_detect_duplicates_bool
341 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

```
342 \bool_new:N \l_@@_initial_open_bool
343 \bool_new:N \l_@@_final_open_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.

```
344 \dim_new:N \l_@@_tabular_width_dim
```

`\l_@@_rule_width_before_dim` will be used before the construction of the array (that is to say during the definition of new types of rules and during the instructions used by the final user in order to require rules of several types).

```
345 \dim_new:N \l_@@_rule_width_before_dim
```

`\l_@@_rule_width_before_dim` will be used *after* the construction of the array (that is to say when we are actually drawing the rules).

```
346 \dim_new:N \l_@@_rule_width_after_dim
```

We use two variables only for legibility. We could use the same since we are not at all at the same moment in the compilation.

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

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

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

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

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

```
349 \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.

```
350 \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`).

```
351 \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`.

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

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

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

```
354 \bool_new:N \l_@@_no_cell_nodes_bool
```

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

```
355 \tl_new:N \g_@@_aux_tl
```

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

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

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

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

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

```
359 \tl_new:N \g_@@_right_delim_tl
```

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

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

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

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

For `\multicolumn`.

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

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

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

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

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

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

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

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

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

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

```
369 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
370 {
```

```
371   \if_mode_math: \else:
```

```
372     \@@_fatal:n { Outside-math-mode }
```

```
373   \fi:
```

```
374 }
```

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.

```
375 \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”.

```
376 \colorlet { nicematrix-last-col } { . }
377 \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`).

```
378 \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*.

```
379 \str_new:N \g_@@_com_or_env_str
380 \str_gset:Nn \g_@@_com_or_env_str { environment }
```

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

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

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

`\g_@@_col_with_trees_clist` is for `draw-trees-in-col`

```
383 \clist_new:N \g_@@_col_with_trees_clist
```

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

```
384 \cs_new:Npn \@@_full_name_env:
385 {
386   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
387   { command \space \c_backslash_str \g_@@_name_env_str }
388   { environment \space \{ \g_@@_name_env_str \} }
389 }
```

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

The argument is given by curryfication.

```
391 \cs_new_protected:Npn \@@_put_in_cell_after_hook:n
392 { \tl_gput_right:Nn \g_@@_cell_after_hook_tl }
```

The so-called `\CodeBefore` is split in two parts because we want to control the order of execution of some instructions.

```
393 \tl_new:N \g_@@_pre_code_before_tl
394 \tl_new:N \g_nicematrix_code_before_tl
```

The value of the key `code-before` will be added to the left of `\g_@@_pre_code_before_tl`. Idem for the code between `\CodeBefore` and `\Body`.

`\g_@@_rules_tl` will contain instructions to draw rules specified by:

- the keys `hlines` and `vlines` (and `hvlines`, `hvlines-except-borders`);
- the specifier `|` in the preamble of the argument;
- the commands `\Hline`;
- the key `hlines`, `vlines` and `hvlines` of a block;
- the key `draw` of a block;

- the command `\diagbox`.

```
395 \tl_new:N \g_@@_rules_tl
```

The so-called `\CodeAfter` is split in two parts because we want to control the order of execution of some instructions.

```
396 \tl_new:N \g_@@_pre_code_after_tl
397 \tl_new:N \g_nicematrix_code_after_tl
```

The `\CodeAfter` provided by the final user (with the key `code-after` or the keyword `\CodeAfter`) will be stored in the second token list.

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

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

```
399 \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.

```
400 \int_new:N \l_@@_old_iRow_int
401 \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).

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

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

```
403 \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.

```
404 \bool_new:N \l_@@_X_columns_aux_bool
405 \dim_new:N \l_@@_X_columns_dim
```

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

```
406 \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).

```
407 \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).

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

```
409 \tl_new:N \l_@@_code_before_tl
410 \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).

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

The following dimensions will be used when drawing the dotted lines but also for the rules.

```
412 \dim_new:N \l_@@_x_initial_dim
413 \dim_new:N \l_@@_y_initial_dim
414 \dim_new:N \l_@@_x_final_dim
415 \dim_new:N \l_@@_y_final_dim

416 \dim_new:N \g_@@_dp_row_zero_dim
417 \dim_new:N \g_@@_ht_row_zero_dim
418 \dim_new:N \g_@@_ht_row_one_dim
419 \dim_new:N \g_@@_dp_ante_last_row_dim
420 \dim_new:N \g_@@_ht_last_row_dim
421 \dim_new:N \g_@@_dp_last_row_dim
```

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

```
422 \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”.

```
423 \dim_new:N \g_@@_width_last_col_dim
424 \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.

```
425 \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!).

```
426 \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.

```
427 \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}`.

```
428 \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.

```
429 \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`).

```
430 \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`).

```
431 \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.

```
432 \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).

```
433 \seq_new:N \g_@@_multicolumn_cells_seq
434 \seq_new:N \g_@@_multicolumn_sizes_seq
```

The following counter will be used for structures such as `\Hline\Hline`.

```
435 \int_new:N \l_@@_multiplicity_int
436 \int_set:Nn \l_@@_multiplicity_int { 1 }
```

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.

```
437 \int_new:N \g_@@_ddots_int
438 \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.

```
439 \dim_new:N \g_@@_delta_x_one_dim
440 \dim_new:N \g_@@_delta_y_one_dim
441 \dim_new:N \g_@@_delta_x_two_dim
442 \dim_new:N \g_@@_delta_y_two_dim
```

The following counters will be used when searching the extremities of a dotted line (we need these counters because of the potential “open” lines in the `\SubMatrix`—the `\SubMatrix` in the code-before).

```
443 \int_new:N \l_@@_row_min_int
444 \int_new:N \l_@@_row_max_int
445 \int_new:N \l_@@_col_min_int
446 \int_new:N \l_@@_col_max_int

447 \int_new:N \l_@@_initial_i_int
448 \int_new:N \l_@@_initial_j_int
449 \int_new:N \l_@@_final_i_int
450 \int_new:N \l_@@_final_j_int
```

The following counters will be used when drawing the rules.

```
451 \int_new:N \l_@@_segment_start_int
452 \int_new:N \l_@@_segment_end_int
```

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

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.

```
453 \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).

```
454 \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`.

```
455 \tl_new:N \l_@@_draw_tl
```

```
456 \seq_new:N \l_@@_tikz_seq
```

```
457 \tl_new:N \l_@@_tikz_rule_tl
```

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 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`).

```
458 \str_new:N \l_@@_hpos_block_str
```

```
459 \str_set:Nn \l_@@_hpos_block_str { c }
```

```
460 \bool_new:N \l_@@_hpos_of_block_cap_bool
```

```
461 \bool_new:N \l_@@_p_block_bool
```

```
462 \bool_new:N \l_@@_fix_vertex_bool
```

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

```
463 \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).

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

The blocks which use the key `-` will store their content in a box. These boxes are numbered with the following counter.

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

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}`).

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

When `\l_@@_dotted_bool` is `true`, a dotted line (with our system) will be drawn.

```
467 \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`).

```
468 \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.

```
469 \int_new:N \l_@@_first_row_int
470 \int_set:Nn \l_@@_first_row_int 1
```

- **First column**

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

```
471 \int_new:N \l_@@_first_col_int
472 \int_set:Nn \l_@@_first_col_int 1
```

- **Last row**

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

```
473 \int_new:N \l_@@_last_row_int
474 \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

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

Idem for `\l_@@_last_col_without_value_bool`

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

- **Last column**

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

```
477 \int_new:N \l_@@_last_col_int
478 \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}
```

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.

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.

```
479 \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`).

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

Some utilities

```
481 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
482 {
```

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

```
483 \def \l_tmpa_tl { #1 }
484 \def \l_tmpb_tl { #2 }
485 }
```

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`.

```
486 \cs_new_protected:Npn \@@_expand_clist_hvlines:NN #1 #2
487 {
488   \clist_if_in:NnF #1 { all }
489   {
490     \clist_clear:N \l_tmpa_clist
491     \clist_map_inline:Nn #1
492     {
493       \tl_if_head_eq_meaning:nNTF { ##1 } -
494       {
```

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.

```
495       \int_if_zero:nF { #2 }
496       {
497         \clist_put_right:Ne \l_tmpa_clist
498         { \int_eval:n { #2 + (##1) + 1 } }
499       }
500     }
501   }
```

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

```
502       \tl_if_in:nnTF { ##1 } { - }
503       { \@@_cut_on_hyphen:w ##1 \q_stop }
504       {
```

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

```
505       \def \l_tmpa_tl { ##1 }
506       \def \l_tmpb_tl { ##1 }
507     }
508     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
509     { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
510   }
511 }
512 \tl_set_eq:NN #1 \l_tmpa_clist
513 }
514 }
```

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.

```

515 \AtBeginDocument
516 {
517     \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
518     \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
519 }

```

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`).
 - 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.

```

520 \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`.

```

521 \int_new:N \g_@@_tabularnote_int
522 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }

```

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

```

523 \seq_new:N \g_@@_notes_seq
524 \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.

```

525 \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.

```

526 \seq_new:N \l_@@_notes_labels_seq
527 \newcounter { nicematrix_draft }
528 \cs_new_protected:Npn \@@_notes_format:n #1
529 {
530   \setcounter { nicematrix_draft } { #1 }
531   \@@_notes_style:n { nicematrix_draft }
532 }

```

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

```

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

```

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

```

534 \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`.

```

535 \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`.

```

536 \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`).

```

537 \AtBeginDocument
538 {
539   \IfPackageLoadedTF { enumitem }
540   {

```

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.

```

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

```

```

556         itemjoin = \quad ,
557         label =
558             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
559     }

```

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`.

```

560     \NewDocumentCommand { \tabularnote } { o m }
561     {
562         \bool_lazy_or:nnT \l_@@_in_env_bool { \cs_if_exist_p:N \@capytype }
563         {
564             \bool_lazy_and:nnTF \l_@@_in_env_bool { ! \l_@@_tabular_bool }
565             { \@@_error:n { tabularnote~forbidden } }
566             {

```

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

```

567                 \bool_if:NTF \l_@@_in_caption_bool
568                 {
569                     \tl_if_novalue:nTF { #1 }
570                     { \@@_tabularnote_caption:nn { #1 } }
571                     { \@@_tabularnote_caption:nn { \exp_not:n { #1 } } }
572                 }
573                 {
574                     \tl_if_novalue:nTF { #1 }
575                     { \@@_tabularnote:nn { #1 } }
576                     { \@@_tabularnote:nn { \exp_not:n { #1 } } }
577                 }
578                 { #2 }
579             }
580         }
581     }
582 }
583 {
584     \NewDocumentCommand \tabularnote { o m }
585     { \@@_err_enumitem_not_loaded: }
586 }
587 }
588 \cs_new_protected:Npn \@@_err_enumitem_not_loaded:
589 {
590     \@@_error_or_warning:n { enumitem~not~loaded }
591     \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
592 }
593 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
594 { \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`.

```

595 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
596 {

```

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).

```

597     \int_zero:N \l_tmpa_int
598     \bool_if:NT \l_@@_notes_detect_duplicates_bool
599     {

```

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

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

If the user have used `\tabulernote` 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@tabulernote`.

```

600      \int_zero:N \l_tmpb_int
601      \seq_map_indexed_inline:Nn \g_@@_notes_seq
602      {
603          \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
604          \tl_if_eq:nnT { { #1 } { #2 } } { { ##2 }
605              {
606                  \tl_if_novalue:nTF { #1 }
607                      { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
608                      { \int_set:Nn \l_tmpa_int { ##1 } }
609                  \seq_map_break:
610              }
611          }
612      \int_if_zero:nF \l_tmpa_int
613      { \int_add:Nn \l_tmpa_int { \g_@@_notes_caption_int } }
614  }
615  \int_if_zero:nT \l_tmpa_int
616  {
617      \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
618      \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabulernote }
619  }
620  \seq_put_right:Ne \l_@@_notes_labels_seq
621  {
622      \tl_if_novalue:nTF { #1 }
623      {
624          \@@_notes_format:n
625          {
626              \int_eval:n
627              {
628                  \int_if_zero:nTF \l_tmpa_int
629                      \c@tabulernote
630                      \l_tmpa_int
631              }
632          }
633      }
634      { #1 }
635  }
636  \peek_meaning:NF \tabulernote
637  {

```

If the following token is *not* a `\tabulernote`, we have finished the sequence of successive commands `\tabulernote` 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`.

```

638      \hbox_set:Nn \l_tmpa_box
639      {

```

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

```

640          \@@_notes_label_in_tabular:n
641          { \seq_use:Nn \l_@@_notes_labels_seq { , } }
642      }

```

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

```

643      \int_gdecr:N \c@tabulernote
644      \int_set_eq:NN \l_tmpa_int \c@tabulernote

```

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

```

645      \int_gincr:N \g_@@_tabularnote_int
646      \refstepcounter { tabularnote }
647      \int_compare:nNnT \l_tmpa_int = \c@tabularnote
648        { \int_gincr:N \c@tabularnote }
649      \seq_clear:N \l_@@_notes_labels_seq
650      \bool_lazy_or:nnTF
651        { \str_if_eq_p:ee c \l_@@_hpos_cell_tl }
652        { \str_if_eq_p:ee r \l_@@_hpos_cell_tl }
653        {
654          \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).

```

655      \skip_horizontal:n { \box_wd:N \l_tmpa_box }
656    }
657    { \box_use:N \l_tmpa_box }
658  }
659 }

```

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`.

```

660 \cs_new_protected:Npn \g_@@_tabularnote_caption:nn #1 #2
661 {
662   \bool_if:NTF \g_@@_caption_finished_bool
663   {
664     \int_compare:nNnT \c@tabularnote = \g_@@_notes_caption_int
665     { \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`!

```

666     \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
667     { \@@_error:n { Identical-notes-in-caption } }
668   }
669   {

```

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

```

670     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
671     {

```

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.

```

672       \bool_gset_true:N \g_@@_caption_finished_bool
673       \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
674       \int_gzero:N \c@tabularnote
675     }
676     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
677   }

```

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!

```

678   \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
679   \seq_put_right:Ne \l_@@_notes_labels_seq
680   {
681     \tl_if_novalue:nTF { #1 }

```



```

682         { \@@_notes_format:n { \int_use:N \c@tabularnote } }
683         { #1 }
684     }
685     \peek_meaning:NF \tabularnote
686     {
687         \@@_notes_label_in_tabular:n { \seq_use:Nn \l_@@_notes_labels_seq { , } }
688         \seq_clear:N \l_@@_notes_labels_seq
689     }
690 }
691 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
692 { \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.

```

693 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
694 {
695     \begin { pgfscope }
696     \pgfset
697     {
698         inner~sep = \c_zero_dim ,
699         minimum~size = \c_zero_dim
700     }
701     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
702     \pgfnode
703     { rectangle }
704     { center }
705     {
706         \vbox_to_ht:nn
707         { \dim_abs:n { #5 - #3 } }
708         {
709             \vfill
710             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { } { }
711         }
712     }
713     { #1 }
714     { }
715     \end { pgfscope }
716 }

```

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

```

717 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
718 {
719     \begin { pgfscope }
720     \pgfset
721     {
722         inner~sep = \c_zero_dim ,
723         minimum~size = \c_zero_dim
724     }
725     \pgftransformshift { \pgfpoint scale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
726     \pgfpointdiff { #3 } { #2 }
727     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
728     \pgfnode
729     { rectangle }
730     { center }

```

```

731     {
732         \vbox_to_ht:nn
733         { \dim_abs:n \l_tmpb_dim }
734         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
735     }
736     { #1 }
737     { }
738 \end { pgfscope }
739 }

```

7 The options

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).

```

740 \bool_new:N \l_@@_caption_above_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.

```

741 \dim_new:N \l_@@_xdots_inter_dim
742 \AtBeginDocument
743 { \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).

```

744 \dim_new:N \l_@@_xdots_shorten_start_dim
745 \dim_new:N \l_@@_xdots_shorten_end_dim
746 \AtBeginDocument
747 {
748     \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
749     \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
750 }

```

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.

```

751 \dim_new:N \l_@@_xdots_radius_dim
752 \AtBeginDocument
753 { \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.

```

754 \tl_new:N \l_@@_xdots_line_style_tl
755 \tl_const:Nn \c_@@_standard_tl { standard }
756 \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`.

```
757 \bool_new:N \l_@@_light_syntax_bool
758 \bool_new:N \l_@@_light_syntax_expanded_bool
```

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

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

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

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

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

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

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

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

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

```
763 \bool_new:N \l_@@_medium_nodes_bool
764 \bool_new:N \l_@@_large_nodes_bool
```

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

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

```
766 \keys_define:nn { nicematrix / xdots }
767 {
```

When the key `xdots/nullify` is used, the instructions like `\vdots` are inserted within a `\hphantom` (and so the constructed matrix has exactly the same size as a matrix constructed with the classical `{matrix}` and `\ldots`, `\vdots`, etc.).

```
768 nullify .bool_set:N = \l_@@_nullify_dots_bool ,
769 brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
770 brace-shift+ .code:n =
771   \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
772 brace-shift+ .value_required:n = true ,
773 brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
774 Vbrace .bool_set:N = \l_@@_Vbrace_bool ,
775 shorten-start .code:n =
776   \AtBeginDocument { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
777 shorten-start .value_required:n = true ,
778 shorten-start+ .code:n =
779   \AtBeginDocument { \dim_add:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
780 shorten-start~+ .meta:n = { shorten-start += #1 } ,
781 shorten-start+ .value_required:n = true ,
782 shorten-end .code:n =
783   \AtBeginDocument { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
784 shorten-end .value_required:n = true ,
785 shorten-end+ .code:n =
786   \AtBeginDocument { \dim_add:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
787 shorten-end+ .value_required:n = true ,
788 shorten-end~+ .meta:n = { shorten-end += #1 } ,
789 shorten .code:n =
```

```

790 \AtBeginDocument
791 {
792     \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
793     \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
794 } ,
795 shorten .value_required:n = true ,
796 shorten+ .code:n =
797     \AtBeginDocument
798     {
799         \dim_add:Nn \l_@@_xdots_shorten_start_dim { #1 }
800         \dim_add:Nn \l_@@_xdots_shorten_end_dim { #1 }
801     } ,
802 shorten~+ .meta:n = { shorten+ = #1 } ,
803 horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
804 horizontal-label .bool_set:N = \l_@@_xdots_h_labels_bool ,
805 line-style .code:n =
806     {
807         \bool_lazy_or:nnTF
808             { \cs_if_exist_p:N \tikzpicture }
809             { \str_if_eq_p:nn { #1 } { standard } }
810             { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
811             { \@@_error:n { bad-option-for-line-style } }
812     } ,
813 line-style .value_required:n = true ,
814 color .tl_set:N = \l_@@_xdots_color_tl ,
815 color .value_required:n = true ,
816 radius .code:n =
817     \AtBeginDocument { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
818 radius .value_required:n = true ,
819 inter .code:n =
820     \AtBeginDocument { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
821 radius .value_required:n = true ,

```

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

```

822 down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
823 up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
824 middle .code:n = \tl_put_right:Nn \l_@@_xdots_middle_tl { #1 } ,

```

The key `draw-first`, which is meant to be used only with `\Ddots` and `\Iddots`, will be caught when `\Ddots` or `\Iddots` is used (during the construction of the array and not when we draw the dotted lines).

```

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

```

```

829 \keys_define:nn { nicematrix / trees }
830 {
831     color.tl_set:N = \l_@@_trees_color_tl ,
832     color .value_required:n = true ,
833     line-width .dim_set:N = \l_@@_trees_line_width_dim ,
834     rounded-corners .dim_set:N = \l_@@_trees_rounded_corners_dim ,
835     rounded-corners .default:n = 2 pt ,
836     unknown .code:n =
837         \@@_unknown_key:nn { nicematrix / trees } { Unknown~key~for~trees }
838 }

```

```

839 \keys_define:nn { nicematrix / rules }
840 {
841     fix-vertex .code:n =

```

```

842     \bool_set_true:N \l_@@_fix_vertex_bool
843     \socket_assign_plug:nn { nicematrix / draw-at-odd-vertex-h } { active }
844     \socket_assign_plug:nn { nicematrix / draw-at-odd-vertex-v } { active } ,
845     color .tl_set:N = \l_@@_rules_color_tl ,
846     color .value_required:n = true ,
847     width .dim_set:N = \arrayrulewidth ,
848     unknown .code:n = \@@_error:n { Unknown~key~for~rules }
849 }

```

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

```

850 \keys_define:nn { nicematrix / Global }
851 {
852     fix-vertex .value_forbidden:n = true ,
853     brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
854     brace-shift+ .code:n =
855         \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
856     brace-shift+ .value_required:n = true ,
857     brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
858     caption-above .code:n = \@@_error_or_warning:n { caption-above-in-env } ,
859     show-cell-names .code = \@@_error_or_warning:n { show-cell-names } ,
860     color-inside .code:n = \@@_fatal:n { key~color~inside } ,
861     colortbl-like .code:n = \@@_fatal:n { key~color~inside } ,
862     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
863     &-in-blocks .meta:n = ampersand-in-blocks ,
864     no-cell-nodes .code:n =
865         \bool_set_true:N \l_@@_no_cell_nodes_bool
866         \cs_set_protected:Npn \@@_node_cell:
867             { \set@color \box_use_drop:N \l_@@_cell_box } ,
868     no-cell-nodes .value_forbidden:n = true ,

```

When the key `rounded-corners` is used, a clipping is applied in the `\CodeBefore` of the environment in order to have rounded corners for the potential colored panels.

```

869     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
870     rounded-corners .default:n = 4 pt ,
871     custom-line .code:n = \@@_custom_line:n { #1 } ,
872     default-line .code:n = \@@_default_line:n { #1 } ,
873     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
874     rules .value_required:n = true ,
875     trees .code:n = \keys_set:nn { nicematrix / trees } { #1 } ,
876     trees .value_required:n = true ,

```

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 `standard-cline`.

```

877     standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
878     cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
879     cell-space-top-limit+ .code:n =
880         \dim_add:Nn \l_@@_cell_space_top_limit_dim { #1 } ,
881     cell-space-top-limit+ .value_required:n = true ,
882     cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
883     cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
884     cell-space-bottom-limit+ .code:n =
885         \dim_add:Nn \l_@@_cell_space_bottom_limit_dim { #1 } ,
886     cell-space-bottom-limit+ .value_required:n = true ,
887     cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
888     cell-space-limits .meta:n =
889         {
890             cell-space-top-limit = #1 ,
891             cell-space-bottom-limit = #1 ,
892         } ,
893     cell-space-limits .value_required:n = true ,
894     cell-space-limits+ .meta:n =

```

```

895     {
896         cell-space-top-limit += #1 ,
897         cell-space-bottom-limit += #1 ,
898     } ,
899     cell-space-limits+ .value_required:n = true ,
900     cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
901     xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
902     light-syntax .code:n =
903         \bool_set_true:N \l_@@_light_syntax_bool
904         \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
905     light-syntax .value_forbidden:n = true ,
906     light-syntax-expanded .code:n =
907         \bool_set_true:N \l_@@_light_syntax_bool
908         \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
909     light-syntax-expanded .value_forbidden:n = true ,

```

The key `end-of-row` specifies the symbol used to mark the ends of rows when the light syntax is used.

```

910     end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
911     end-of-row .value_required:n = true ,
912     end-of-row .initial:n = ; ,
913
914     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
915     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
916     last-row .int_set:N = \l_@@_last_row_int ,
917     last-row .default:n = -1 ,
918
919     code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
920     code-for-first-col .value_required:n = true ,
921     code-for-first-col+ .code:n =
922         { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
923     code-for-first-col+ .value_required:n = true ,
924     code-for-first-col~+ .meta:n = { code-for-first-col+ = #1 } ,
925
926     code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
927     code-for-last-col .value_required:n = true ,
928     code-for-last-col+ .code:n =
929         { \tl_put_right:Nn \l_@@_code_for_last_col_tl { #1 } } ,
930     code-for-last-col+ .value_required:n = true ,
931     code-for-last-col~+ .meta:n = { code-for-last-col+ = #1 } ,
932
933     code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
934     code-for-first-row .value_required:n = true ,
935     code-for-first-row+ .code:n =
936         { \tl_put_right:Nn \l_@@_code_for_first_row_tl { #1 } } ,
937     code-for-first-row+ .value_required:n = true ,
938     code-for-first-row~+ .meta:n = { code-for-first-row+ = #1 } ,
939
940     code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
941     code-for-last-row .value_required:n = true ,
942     code-for-last-row+ .code:n =
943         { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
944     code-for-last-row+ .value_required:n = true ,
945     code-for-last-row~+ .meta:n = { code-for-last-row+ = #1 } ,
946
947     hlines .clist_set:N = \l_@@_hlines_clist ,
948     hlines .default:n = all ,
949     vlines .clist_set:N = \l_@@_vlines_clist ,
950     vlines .default:n = all ,
951
952     vlines-in-sub-matrix .code:n =
953     {
954         \tl_if_single_token:nTF { #1 }
955         {

```

```

956      \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
957      { \@@_error:nn { Forbidden~letter } { #1 } }

```

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

```

958      { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
959    }
960    { \@@_error:n { One~letter~allowed } }
961  } ,
962  vlines-in-sub-matrix .value_required:n = true ,
963  hvlines .code:n =
964  {
965    \bool_set_true:N \l_@@_hvlines_bool
966    \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
967    \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
968  } ,
969  hvlines .value_forbidden:n = true ,
970  hvlines-except-borders .code:n =
971  {
972    \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
973    \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
974    \bool_set_true:N \l_@@_hvlines_bool
975    \bool_set_true:N \l_@@_except_borders_bool
976  } ,
977  hvlines-except-borders .value_forbidden:n = true ,
978  parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,
979  parallelize-diags .initial:n = true ,

```

With the option `renew-dots`, the command `\cdots`, `\ldots`, `\vdots`, `\ddots`, etc. are redefined and behave like the commands `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots`, etc.

```

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

```

The code of the key `pgf-node-code` will be used when the nodes of the cells (that is to say the nodes of the form `i-j`) will be created.

```

1001  pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
1002  pgf-node-code .value_required:n = true ,
1003  }

```

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

```

1004  \keys_define:nn { nicematrix / environments }
1005  {
1006    draw-trees-in-col .code:n =
1007    \clist_gput_right:Nn \g_@@_col_with_trees_clist { #1 } ,

```

```

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

```

The options `c`, `t` and `b` of the environment `{NiceArray}` have the same meaning as the option of the classical environment `{array}`.

```

1023 c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
1024 t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
1025 b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,

```

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).

```

1026 baseline .tl_set:N = \l_@@_baseline_tl ,
1027 baseline .value_required:n = true ,
1028 baseline .initial:n = c ,
1029 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)`.

```

1030 \str_if_eq:eeTF { #1 } { auto }
1031 { \bool_set_true:N \l_@@_auto_columns_width_bool }
1032 { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
1033 columns-width .value_required:n = true ,
1034 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.

```

1035 \legacy_if:nF { measuring@ }
1036 {
1037   \str_set:Nn \l_@@_name_str { #1 }
1038   \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
1039   { \@@_err_duplicate_names:n { #1 } }
1040   { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
1041 } ,
1042 name .value_required:n = true ,
1043 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
1044 code-after .value_required:n = true ,
1045 }
1046 \cs_set:Npn \@@_err_duplicate_names:n #1
1047 { \@@_error:nn { Duplicate-name } { #1 } }
1048 \keys_define:nn { nicematrix / notes }
1049 {
1050   no-print .bool_set:N = \l_@@_notes_no_print_bool ,
1051   para .bool_set:N = \l_@@_notes_para_bool ,
1052   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
1053   code-before .value_required:n = true ,
1054   code-before+ .code:n =
1055     \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1056   code-before+ .value_required:n = true ,
1057   code-before++ .code:n =

```



```

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

```

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

```

1091 \keys_define:nn { nicematrix / delimiters }
1092 {
1093     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1094     color .tl_set:N = \l_@@_delimiters_color_tl ,
1095     color .value_required:n = true ,
1096 }

```

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

```

1097 \keys_define:nn { nicematrix }
1098 {
1099     NiceMatrixOptions .inherit:n =
1100     { nicematrix / Global } ,
1101     NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1102     NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1103     NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1104     NiceMatrixOptions / trees .inherit:n = nicematrix / trees ,
1105     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1106     SubMatrix / rules .inherit:n = nicematrix / rules ,
1107     CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1108     CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,

```

```

1109 CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1110 NiceMatrix .inherit:n =
1111 {
1112     nicematrix / Global ,
1113     nicematrix / environments ,
1114 } ,
1115 NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1116 NiceMatrix / rules .inherit:n = nicematrix / rules ,
1117 NiceMatrix / trees .inherit:n = nicematrix / trees ,
1118 NiceTabular .inherit:n =
1119 {
1120     nicematrix / Global ,
1121     nicematrix / environments
1122 } ,
1123 NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1124 NiceTabular / rules .inherit:n = nicematrix / rules ,
1125 NiceTabular / notes .inherit:n = nicematrix / notes ,
1126 NiceTabular / trees .inherit:n = nicematrix / trees ,
1127 NiceArray .inherit:n =
1128 {
1129     nicematrix / Global ,
1130     nicematrix / environments ,
1131 } ,
1132 NiceArray / xdots .inherit:n = nicematrix / xdots ,
1133 NiceArray / rules .inherit:n = nicematrix / rules ,
1134 NiceArray / trees .inherit:n = nicematrix / trees ,
1135 pNiceArray .inherit:n =
1136 {
1137     nicematrix / Global ,
1138     nicematrix / environments ,
1139 } ,
1140 pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1141 pNiceArray / rules .inherit:n = nicematrix / rules ,
1142 pNiceArray / trees .inherit:n = nicematrix / trees
1143 }

```

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

```

1144 \keys_define:nn { nicematrix / NiceMatrixOptions }
1145 {
1146     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1147     delimiters / color .value_required:n = true ,
1148     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1149     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1150     delimiters .value_required:n = true ,
1151     width .dim_set:N = \l_@@_width_dim ,
1152     last-col .code:n =
1153     \tl_if_empty:nF { #1 }
1154     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1155     \int_zero:N \l_@@_last_col_int ,
1156     small .bool_set:N = \l_@@_small_bool ,
1157     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.

```

1158 renew-matrix .code:n = \@@_renew_matrix: ,
1159 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}`.

```

1160 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.

```
1161     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)`.

```
1162     \str_if_eq:eeTF { #1 } { auto }
1163     { \@@_error:n { Option~auto~for~columns-width } }
1164     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
```

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

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

`\NiceMatrixOptions` is the command of the `nicematrix` package to fix options at the document level. The scope of these specifications is the current TeX group.

```
1180 \NewDocumentCommand \NiceMatrixOptions { m }
1181 { \keys_set:nn { nicematrix / NiceMatrixOptions } { #1 } }
```

We finalise the definition of the set of keys “`nicematrix / NiceMatrix`”. That set of keys will be used by `{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.

```
1182 \keys_define:nn { nicematrix / NiceMatrix }
1183 {
1184     last-col .code:n = \tl_if_empty:nTF { #1 }
1185     {
1186         \bool_set_true:N \l_@@_last_col_without_value_bool
1187         \int_set:Nn \l_@@_last_col_int { -1 }
1188     }
1189     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1190     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1191     columns-type .value_required:n = true ,
1192     l .meta:n = { columns-type = l } ,
1193     r .meta:n = { columns-type = r } ,
1194     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1195     delimiters / color .value_required:n = true ,
1196     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1197     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1198     delimiters .value_required:n = true ,
1199     small .bool_set:N = \l_@@_small_bool ,
1200     small .value_forbidden:n = true ,
1201     unknown .code:n =
1202     \@@_unknown_key:nn
1203     { nicematrix / Global , nicematrix / environments , nicematrix / NiceMatrix }
1204     { Unknown-key-for-NiceMatrix }
1205 }
```

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

```
1206 \keys_define:nn { nicematrix / NiceArray }
1207 {
```

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.

```
1208   small .bool_set:N = \l_@@_small_bool ,
1209   small .value_forbidden:n = true ,
1210   last-col .code:n = \tl_if_empty:nF { #1 }
1211     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1212     \int_zero:N \l_@@_last_col_int ,
1213   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1214   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1215   unknown .code:n =
1216     \@@_unknown_key:nn
1217     { nicematrix / NiceArray , nicematrix / Global , nicematrix / environments }
1218     { Unknown-key-for-NiceArray }
1219 }

1220 \keys_define:nn { nicematrix / pNiceArray }
1221 {
1222   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1223   last-col .code:n = \tl_if_empty:nF { #1 }
1224     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1225     \int_zero:N \l_@@_last_col_int ,
1226   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1227   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1228   delimiters / color .value_required:n = true ,
1229   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1230   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1231   delimiters .value_required:n = true ,
1232   small .bool_set:N = \l_@@_small_bool ,
1233   small .value_forbidden:n = true ,
1234   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1235   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1236   unknown .code:n =
1237     \@@_unknown_key:nn
1238     { nicematrix / pNiceArray , nicematrix / Global , nicematrix / environments }
1239     { Unknown-key-for-NiceMatrix }
1240 }
```

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

```
1241 \keys_define:nn { nicematrix / NiceTabular }
1242 {
```

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.

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

```

1257         \int_zero:N \l_@@_last_col_int ,
1258     r .code:n = \@@_error:n { r-or~l-with-preamble } ,
1259     l .code:n = \@@_error:n { r-or~l-with-preamble } ,
1260     unknown .code:n =
1261         \@@_unknown_key:nn
1262         { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
1263         { Unknown-key-for-NiceTabular }
1264 }

```

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
```

```

1265 \keys_define:nn { nicematrix / CodeAfter }
1266 {
1267     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1268     delimiters / color .value_required:n = true ,
1269     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1270     rules .value_required:n = true ,
1271     xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1272     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1273     sub-matrix .value_required:n = true ,
1274     unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1275 }

```

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}`).

```

1276 \cs_new_protected:Npn \@@_cell_begin:
1277 {

```

`\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.

```
1278     \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).

```
1279     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1280     \cs_set_eq:NN \Hline \@@_Hline_in_cell:
```

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

```
1281     \int_gincr:N \c@jCol
```

Now, we increment the counter of the rows. We don't do this incrementation in the `\everycr` because some packages, like `arydshln`, create special rows in the `\halign` that we don't want to take into account.

Here is a version with the standard syntax of L3.

```

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

```

We will use a version a little more efficient.

```

1282 \if_int_compare:w \c@jCol = \c_one_int
1283 \if_int_compare:w \l_@@_first_col_int = \c_one_int
1284 \@@_begin_of_row:
1285 \fi:
1286 \fi:

```

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

```

1287 \hbox_set:Nw \l_@@_cell_box

```

The following command is nullified in the tabulars.

```

1288 \@@_tuning_not_tabular_begin:
1289 \@@_tuning_exterior_rows:
1290 \g_@@_row_style_tl
1291 }
1292 \cs_new_protected:Npn \@@_tuning_exterior_rows: { }

```

Here is a version with the standard syntax of L3.

```

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

```

We will use a version a little more efficient.

```

1293 \cs_new_protected:Npn \@@_tuning_first_last_row:
1294 {
1295 \if_int_compare:w \c@iRow = \c_zero_int
1296 \if_int_compare:w \c@jCol > \c_zero_int
1297 \l_@@_code_for_first_row_tl
1298 \xglobal \colorlet { nicematrix-first-row } { . }
1299 \fi:
1300 \else:
1301 \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row:
1302 \fi:
1303 }

```

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

```

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

```

1304 \cs_new_protected:Npn \@@_tuning_last_row:
1305 {
1306 \if_int_compare:w \c@iRow = \l_@@_last_row_int
1307 \l_@@_code_for_last_row_tl
1308 \xglobal \colorlet { nicematrix-last-row } { . }
1309 \fi:
1310 }

```

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

```
1311 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```
1312 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1313 {
1314   \m@th
1315   $ % $
```

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

```
1316   \@@_tuning_key_small:
1317 }
1318 \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.

```
1319 \cs_new_protected:Npn \@@_begin_of_row:
1320 {
1321   \int_gincr:N \c@iRow
1322   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1323   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1324   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1325   \pgfpicture
1326   \pgfrememberpicturepositiononpagetrue
1327   \pgfcoordinate
1328     { \@@_env: - row - \int_use:N \c@iRow - base }
1329     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1330   \str_if_empty:NF \l_@@_name_str
1331     {
1332       \pgfnodealias
1333         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1334         { \@@_env: - row - \int_use:N \c@iRow - base }
1335     }
1336   \endpgfpicture
1337 }
```

Remark: If the key `create-cell-nodes` of the `\CodeBefore` is used, then we will add some lines to that command.

The following code is used in each cell of the array. It actualises quantities that, at the end of the array, will give information about the vertical dimension of the two first rows and the two last rows. If the user uses the `last-row`, some lines of code will be dynamically added to this command. Here is a version with the standard syntax of L3.

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

We will use a version a little more efficient.

```

1338 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1339 {
1340   \if_int_compare:w \c@iRow = \c_zero_int
1341     \if_dim:w \box_dp:N \l_@@_cell_box > \g_@@_dp_row_zero_dim
1342       \global \g_@@_dp_row_zero_dim = \box_dp:N \l_@@_cell_box
1343     \fi:
1344     \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_zero_dim
1345       \global \g_@@_ht_row_zero_dim = \box_ht:N \l_@@_cell_box
1346     \fi:
1347   \else:
1348     \if_int_compare:w \c@iRow = \c_one_int
1349       \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_one_dim
1350         \global \g_@@_ht_row_one_dim = \box_ht:N \l_@@_cell_box
1351       \fi:
1352     \fi:
1353   \fi:
1354 }

1355 \cs_new_protected:Npn \@@_rotate_cell_box:
1356 {
1357   \box_rotate:Nn \l_@@_cell_box { 90 }
1358   \bool_if:NTF \g_@@_rotate_c_bool
1359   {
1360     \hbox_set:Nn \l_@@_cell_box
1361     {
1362       \m@th
1363       $ % $
1364       \vcenter { \box_use:N \l_@@_cell_box }
1365       $ % $
1366     }
1367   }
1368   {
1369     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1370     {
1371       \vbox_set_top:Nn \l_@@_cell_box
1372       {
1373         \vbox_to_zero:n { }
1374         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1375         \box_use:N \l_@@_cell_box
1376       }
1377     }
1378   }
1379   \bool_gset_false:N \g_@@_rotate_bool
1380   \bool_gset_false:N \g_@@_rotate_c_bool
1381 }

```

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

```

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

```



```

    }
\dim_compare:nNnT { \g_@@_blocks_ht_dim } > { \c_zero_dim }
{
  \dim_compare:nNnT \g_@@_blocks_ht_dim > { \box_ht:N \l_@@_cell_box }
  { \box_set_ht:Nn \l_@@_cell_box \g_@@_blocks_ht_dim }
  \dim_gzero:N \g_@@_blocks_ht_dim
}
}

```

Here is a version slightly more efficient.

```

1382 \cs_set_protected:Npn \@@_adjust_size_box:
1383 {
1384   \if_dim:w \g_@@_blocks_wd_dim > \c_zero_dim
1385     \if_dim:w \g_@@_blocks_wd_dim > \box_wd:N \l_@@_cell_box
1386       \box_wd:N \l_@@_cell_box = \g_@@_blocks_wd_dim
1387     \fi:
1388     \global \g_@@_blocks_wd_dim = \c_zero_dim
1389   \fi:
1390   \if_dim:w \g_@@_blocks_dp_dim > \c_zero_dim
1391     \if_dim:w \g_@@_blocks_dp_dim > \box_dp:N \l_@@_cell_box
1392       \box_dp:N \l_@@_cell_box = \g_@@_blocks_dp_dim
1393     \fi
1394     \global \g_@@_blocks_dp_dim = \c_zero_dim
1395   \fi:
1396   \if_dim:w \g_@@_blocks_ht_dim > \c_zero_dim
1397     \if_dim:w \g_@@_blocks_ht_dim > \box_ht:N \l_@@_cell_box
1398       \box_ht:N \l_@@_cell_box = \g_@@_blocks_ht_dim
1399     \fi:
1400     \global \g_@@_blocks_ht_dim = \c_zero_dim
1401   \fi:
1402 }
1403 \cs_new_protected:Npn \@@_cell_end:
1404 {

```

The following command is nullified in the tabulars.

```

1405   \@@_tuning_not_tabular_end:
1406   \hbox_set_end:
1407   \@@_cell_end_i:
1408 }

```

```

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

```

```

\bool_gset_false:N \g_@@_empty_cell_bool
\bool_gset_false:N \g_@@_not_empty_cell_bool
}

```

Here is a version slightly more efficient.

```

1409 \cs_new_protected:Npn \@@_cell_end_i:
1410 {

```

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.

```

1411 \g_@@_cell_after_hook_tl
1412 \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
1413 \@@_adjust_size_box:
1414 \box_set_ht:Nn \l_@@_cell_box
1415 { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1416 \box_set_dp:Nn \l_@@_cell_box
1417 { \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”).

```

1418 \@@_update_max_cell_width:

```

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

```

1419 \@@_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.

```

1420 \bool_if:NTF \g_@@_empty_cell_bool
1421 { \box_use_drop:N \l_@@_cell_box }
1422 {
1423 \bool_if:NTF \g_@@_not_empty_cell_bool
1424 \@@_print_node_cell:
1425 {
1426 \if_dim:w \box_wd:N \l_@@_cell_box > \c_zero_dim
1427 \@@_print_node_cell:
1428 \else:
1429 \box_use_drop:N \l_@@_cell_box
1430 \fi:
1431 }
1432 }
1433 \if_int_compare:w \c@jCol > \g_@@_col_total_int
1434 \global \g_@@_col_total_int = \c@jCol
1435 \fi:
1436 \global \let \g_@@_empty_cell_bool \c_false_bool
1437 \global \let \g_@@_not_empty_cell_bool \c_false_bool
1438 }

```

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

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

We will use the following version, slightly more efficient:

```
1439 \cs_new_protected:Npn \@@_update_max_cell_width:
1440 {
1441   \if_dim:w \box_wd:N \l_@@_cell_box > \g_@@_max_cell_width_dim
1442     \global \g_@@_max_cell_width_dim = \box_wd:N \l_@@_cell_box
1443   \fi:
1444 }
```

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`).

```
1445 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1446 {
1447   \@@_math_toggle:
1448   \hbox_set_end:
1449   \bool_if:NF \g_@@_rotate_bool
1450   {
1451     \hbox_set:Nn \l_@@_cell_box
1452     {
1453       \makebox [ \l_@@_col_width_dim ] [ s ]
1454       { \hbox_unpack_drop:N \l_@@_cell_box }
1455     }
1456   }
1457   \@@_cell_end_i:
1458 }
```

```
1459 \pgfset
1460 {
1461   nicematrix / cell-node /.style =
1462   {
1463     inner~sep = \c_zero_dim ,
1464     minimum~width = \c_zero_dim
1465   }
1466 }
```

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.

```
1467 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1468 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1469 {
1470   \use:c
1471   {
1472     __siunitx_table_align_
1473     \bool_if:NTF \l__siunitx_table_text_bool
1474       \l__siunitx_table_align_text_tl
1475       \l__siunitx_table_align_number_tl
1476     :n
1477   }
1478   { #1 }
1479 }
```

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!

```

1480 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1481 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1482 {
1483   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1484   \hbox:n
1485   {
1486     \pgfsys@markposition
1487     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1488   }
1489   #1
1490   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1491   \hbox:n
1492   {
1493     \pgfsys@markposition
1494     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1495   }
1496 }

1497 \cs_new_protected:Npn \@@_print_node_cell:
1498 {
1499   \socket_use:nn { nicematrix / siunitx-wrap }
1500   { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1501 }

```

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

```

1502 \cs_new_protected:Npn \@@_node_cell:
1503 {
1504   \pgfpicture
1505   \pgfsetbaseline \c_zero_dim
1506   \pgfrememberpicturepositiononpagetrue
1507   \pgfset { nicematrix / cell-node }
1508   \pgfnode
1509   { rectangle }
1510   { base }
1511   {

```

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

```

1512   \sys_if_engine_xetex:T { \set@color }
1513   \box_use:N \l_@@_cell_box
1514 }
1515 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1516 { \l_@@_pgf_node_code_tl }
1517 \str_if_empty:NF \l_@@_name_str
1518 {
1519   \pgfnodealias
1520   { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1521   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1522 }
1523 \endpgfpicture
1524 }

```

The second argument of the following command `\@@_instruction_of_type:nnn` defined below is the type of the instruction (`Cdots`, `Vdots`, `Ddots`, etc.). The third argument is the list of options. This command writes in the corresponding `\g_@@_type_lines_tl` the instruction which will actually draw the line after the construction of the matrix.

For example, for the following matrix,

```
\begin{pNiceMatrix}
1 & 2 & 3 & 4 \\
5 & \Cdots & & 6 \\
7 & \Cdots & [color=red] & 
\end{pNiceMatrix}
```

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

the content of `\g_@@_Cdots_lines_tl` will be:

```
\@@_draw_Cdots:nnn {2}{2}{ }
\@@_draw_Cdots:nnn {3}{2}{color=red}
```

The first argument is a boolean which indicates whether you must put the instruction on the left or on the right on the list of instructions (with consequences for the parallelisation of the diagonal lines).

```
1525 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1526 {
1527   \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce
1528   { g_@@_ #2 _ lines _ tl }
1529   {
1530     \use:c { @@ _ draw _ #2 : nnn }
1531     { \int_use:N \c@iRow }
1532     { \int_use:N \c@jCol }
1533     { \exp_not:n { #3 } }
1534   }
1535 }

1536 \cs_new_protected:Npn \@@_array:n
1537 {
1538   \dim_set:Nn \col@sep
1539   { \bool_if:nTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1540   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1541   { \def \@halignto { } }
1542   { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }
```

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

```
1543   \@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).
1544   [ \str_if_eq:eeTF c \l_@@_baseline_tl c t ]
1545   }
1546 \cs_generate_variant:Nn \@@_array:n { o }
```

We keep in memory the standard version of `\ar@ialign` because we will redefine `\ar@ialign` in the environment `{NiceArrayWithDelims}` but restore the standard version for use in the cells of the array. 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`.

```
1547 \cs_new_eq:cN { @@_old_ar@ialign: } \ar@ialign
```

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

```
1548 \cs_new_protected:Npn \@@_create_row_node:
1549 {
1550   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1551   {
1552     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1553     \@@_create_row_node_i:
1554   }
1555 }

1556 \cs_new_protected:Npn \@@_create_row_node_i:
1557 {
```

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

```

1558   \hbox
1559   {
1560     \bool_if:NT \l_@@_code_before_bool
1561     {
1562       \vtop
1563       {
1564         \skip_vertical:N 0.5\arrayrulewidth
1565         \pgfsys@markposition
1566         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1567         \skip_vertical:N -0.5\arrayrulewidth
1568       }
1569     }
1570     \pgfpicture
1571     \pgfrememberpicturerepositiononpagetrue
1572     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1573     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1574     \str_if_empty:NF \l_@@_name_str
1575     {
1576       \pgfnodealias
1577       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1578       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1579     }
1580     \endpgfpicture
1581   }
1582 }

```

```

1583 \cs_new_protected:Npn \@@_in_everycr:
1584 {
1585   \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1586   \tbl_update_cell_data_for_next_row:
1587   \int_gzero:N \c@jCol
1588   \bool_gset_false:N \g_@@_after_col_zero_bool
1589   \bool_if:NF \g_@@_row_of_col_done_bool
1590   {
1591     \@@_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).

```

1592   \clist_if_empty:NF \l_@@_hlines_clist
1593   {
1594     \str_if_eq:eeF \l_@@_hlines_clist { all }
1595     {
1596       \clist_if_in:NeT
1597       \l_@@_hlines_clist
1598       { \int_eval:n { \c@iRow + 1 } }
1599     }
1600   }

```

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.

```

1601     \int_compare:nNnT \c@iRow > { -1 }
1602     {
1603       \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1604       { \hrule height \arrayrulewidth width \c_zero_dim }
1605     }
1606   }
1607 }
1608 }
1609 }

```

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

```

1610 \cs_set_protected:Npn \@@_renew_dots:
1611 {
1612   \cs_set_eq:NN \ldots \@@_Ldots:
1613   \cs_set_eq:NN \cdots \@@_Cdots:
1614   \cs_set_eq:NN \vdots \@@_Vdots:
1615   \cs_set_eq:NN \ddots \@@_Ddots:
1616   \cs_set_eq:NN \iddots \@@_Iddots:
1617   \cs_set_eq:NN \dots \@@_Ldots:
1618   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1619 }

```

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

```

1620 \AtBeginDocument
1621 {
1622   \IfPackageLoadedTF { booktabs }
1623   {
1624     \cs_new_protected:Npn \@@_patch_booktabs:
1625     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1626   }
1627   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1628 }

```

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`.

```

1629 \cs_new_protected:Npn \@@_some_initialization:
1630 {
1631   \@@_everycr:
1632   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1633   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1634   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1635   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1636   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1637   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1638 }

```

`\@@_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`.

```

1639 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1640 {

```

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 main blocks will be added in `\g_@@_pos_of_blocks_seq` during the construction of the array.

```

1641   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

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

Idem for other sequences written on the aux file.

```
1642 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1643 \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.

```
1644 \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.

```
1645 \fp_gzero:N \g_@@_total_X_weight_fp
1646 \bool_gset_false:N \g_@@_V_of_X_bool

1647 \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1648 \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol

1649 \@@_patch_booktabs:
1650 \box_clear_new:N \l_@@_cell_box
1651 \normalbaselines
```

If the option `small` is used, we have to do some tuning. In particular, we change the value of `\arraystretch` (this parameter is used in the construction of `\@arstrutbox` in the beginning of `{array}`).

```
1652 \bool_if:NT \l_@@_small_bool
1653 {
1654     \def \arraystretch { 0.47 }
1655     \dim_set:Nn \arraycolsep { 1.45 pt }
```

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

```
1656 \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1657 }
```

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!

```
1658 \bool_if:NT \g_@@_create_cell_nodes_bool
1659 {
1660     \tl_put_right:Nn \@@_begin_of_row:
1661     {
1662         \pgfsys@markposition
1663         { \@@_env: - row - \int_use:N \c@iRow - base }
1664     }
1665     \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1666 }
```

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`.

```
1667 \def \ar@ialign
1668 {
1669     \tbl_init_cell_data_for_table:
1670     \@@_some_initialization:
1671     \dim_zero:N \tabskip
```


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

```

1672     \cs_set_eq:Nc \ar@ialign { @@_old_ar@ialign: }
1673     \halign
1674 }

```

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).

```

1675     \cs_set_eq:NN @@_old_ldots: \ldots
1676     \cs_set_eq:NN @@_old_cdots: \cdots
1677     \cs_set_eq:NN @@_old_vdots: \vdots
1678     \cs_set_eq:NN @@_old_ddots: \ddots
1679     \cs_set_eq:NN @@_old_iddots: \iddots
1680     \bool_if:NTF \l_@@_standard_cline_bool
1681     { \cs_set_eq:NN \cline @@_standard_cline: }
1682     { \cs_set_eq:NN \cline @@_cline: }
1683     \cs_set_eq:NN \Ldots @@_Ldots:
1684     \cs_set_eq:NN \Cdots @@_Cdots:
1685     \cs_set_eq:NN \Vdots @@_Vdots:
1686     \cs_set_eq:NN \Ddots @@_Ddots:
1687     \cs_set_eq:NN \Iddots @@_Iddots:
1688     \cs_set_eq:NN \Hline @@_Hline:
1689     \cs_set_eq:NN \Hspace @@_Hspace:
1690     \cs_set_eq:NN \Hdotsfor @@_Hdotsfor:
1691     \cs_set_eq:NN \Vdotsfor @@_Vdotsfor:
1692     \cs_set_eq:NN \Block @@_Block:
1693     \cs_set_eq:NN \rotate @@_rotate:
1694     \cs_set_eq:NN \OnlyMainNiceMatrix @@_OnlyMainNiceMatrix:n
1695     \cs_set_eq:NN \dotfill @@_dotfill:
1696     \cs_set_eq:NN \CodeAfter @@_CodeAfter:
1697     \cs_if_free:NT \Body { \cs_set_eq:NN \Body @@_Body: }
1698     \cs_if_free:NT \CodeBefore { \cs_set_eq:NN \CodeBefore @@_CodeBefore: }
1699     \cs_set_eq:NN \diagbox @@_diagbox:nn
1700     \cs_set_eq:NN \NotEmpty @@_NotEmpty:
1701     \cs_set_eq:NN \TopRule @@_TopRule
1702     \cs_set_eq:NN \MidRule @@_MidRule
1703     \cs_set_eq:NN \BottomRule @@_BottomRule
1704     \cs_set_eq:NN \RowStyle @@_RowStyle:n
1705     \cs_set_eq:NN \Hbrace @@_Hbrace
1706     \cs_set_eq:NN \Vbrace @@_Vbrace
1707     \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1708     { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1709     \cs_set_eq:NN \cellcolor @@_cellcolor_tabular
1710     \cs_set_eq:NN \rowcolor @@_rowcolor_tabular
1711     \cs_set_eq:NN \rowcolors @@_rowcolors_tabular
1712     \cs_set_eq:NN \rowlistcolors @@_rowlistcolors_tabular
1713     \int_if_zero:nTF \l_@@_first_row_int
1714     { \cs_gset_eq:NN @@_tuning_exterior_rows: @@_tuning_first_last_row: }
1715     {
1716         \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1717         { \cs_gset_eq:NN @@_tuning_exterior_rows: @@_tuning_last_row: }
1718     }
1719     \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:`.

```

1720     \cs_set_eq:NN \multicolumn @@_multicolumn:nnn
1721     \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1722     { \cs_set_eq:NN \multicolumn @@_old_multicolumn: }
1723     @@_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!).

```

1724   \tl_if_exist:NT \l_@@_note_in_caption_tl
1725   {
1726     \tl_if_empty:NF \l_@@_note_in_caption_tl
1727     {
1728       \int_gset:Nn \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1729       \int_gset:Nn \c@tabularnote \l_@@_note_in_caption_tl
1730     }
1731   }

```

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).

```

1732   \seq_gclear:N \g_@@_multicolumn_cells_seq
1733   \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).

```

1734   \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`).

```

1735   \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.

```

1736   \int_gzero:N \g_@@_col_total_int
1737   \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1738   \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.

```

1739   \tl_gclear_new:N \g_@@_Cdots_lines_tl
1740   \tl_gclear_new:N \g_@@_Ldots_lines_tl
1741   \tl_gclear_new:N \g_@@_Vdots_lines_tl
1742   \tl_gclear_new:N \g_@@_Ddots_lines_tl
1743   \tl_gclear_new:N \g_@@_Iddots_lines_tl
1744   \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1745   \tl_gclear:N \g_nicematrix_code_before_tl
1746   \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]`).

```

1747   \dim_zero_new:N \l_@@_left_delim_dim
1748   \dim_zero_new:N \l_@@_right_delim_dim
1749   \bool_if:NTF \g_@@_delims_bool
1750   {

```

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

```

1751     \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1752     \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1753     \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1754     \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1755   }

```

```

1756 {
1757   \dim_gset:Nn \l_@@_left_delim_dim
1758   { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1759   \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1760 }
1761 }

```

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.

```

1762 \cs_new_protected:Npn \@@_pre_array:
1763 {
1764   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1765   \int_gzero_new:N \c@iRow
1766   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1767   \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).

```

1768   \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1769   { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1770   \int_compare:nNnT \l_@@_last_col_int > \c_zero_int
1771   { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1772   \bool_if:NT \g_@@_aux_found_bool
1773   {
1774     \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1775     \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1776     \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1777     \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1778   }

```

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).

```

1779   \int_compare:nNnT \l_@@_last_row_int = { -1 }
1780   {
1781     \bool_set_true:N \l_@@_last_row_without_value_bool
1782     \bool_if:NT \g_@@_aux_found_bool
1783     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1784   }
1785   \int_compare:nNnT \l_@@_last_col_int = { -1 }
1786   {
1787     \bool_if:NT \g_@@_aux_found_bool
1788     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1789   }

```

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”.

```

1790   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1791   {
1792     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1793     {
1794       \dim_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1795       { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1796       \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }

```

```

1797         { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1798     }
1799 }

```

```

1800 \seq_gclear:N \g_@@_cols_vlism_seq
1801 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1802 \bool_if:NT \l_@@_code_before_bool \@@_exec_code_before:

```

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

```

1803 \@@_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).

```

1804 \hbox_set:Nw \l_@@_the_array_box
1805 \skip_horizontal:N \l_@@_left_margin_dim
1806 \skip_horizontal:N \l_@@_extra_left_margin_dim
1807 \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).

```

1808 \m@th
1809 $ % $
1810 \bool_if:NTF \l_@@_light_syntax_bool
1811   { \use:c { @@-light-syntax } }
1812   { \use:c { @@-normal-syntax } }
1813 }

```

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

```

1814 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1815 {
1816   \tl_set:Nn \l_tmpa_tl { #1 }
1817   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1818     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1819   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1820   \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...

```

1821 \@@_pre_array:
1822 }

```

9 The `\CodeBefore`

```

1823 \cs_new_protected_nopar:Npn \@@_Body: { \@@_fatal:n { Body~alone } }
1824 \cs_new_protected_nopar:Npn \@@_CodeBefore: { \@@_fatal:n { Bad~use~of~CodeBefore } }

```

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

```

1825 \cs_new_protected:Npn \@@_pre_code_before:
1826 {

```

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.

```

1827 \pgfsys@markposition { \@@_env: - position }
1828 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1829 \pgfpicture
1830 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1831 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1832 {
1833   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1834   \pgfcoordinate { \@@_env: - row - ##1 }
1835   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1836 }

```

Now, the recreation of the col nodes.

```

1837 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1838 {
1839   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1840   \pgfcoordinate { \@@_env: - col - ##1 }
1841   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1842 }

```

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

```

1843 \bool_if:NT \g_@@_create_cell_nodes_bool \@@_recreate_cell_nodes:
1844 \endpgfpicture

```

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

```

1845 \@@_create_diag_nodes:

```

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

```

1846 \@@_create_blocks_nodes:
1847 \IfPackageLoadedT { tikz }
1848 {
1849   \tikzset
1850   {
1851     every-picture / .style =
1852     { overlay , name-prefix = \@@_env: - }
1853   }
1854 }
1855 \cs_set_eq:NN \cellcolor \@@_cellcolor
1856 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1857 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1858 \cs_set_eq:NN \rowcolor \@@_rowcolor
1859 \cs_set_eq:NN \rowcolors \@@_rowcolors
1860 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1861 \cs_set_eq:NN \arraycolor \@@_arraycolor
1862 \cs_set_eq:NN \columncolor \@@_columncolor
1863 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1864 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1865 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1866 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1867 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1868 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1869 }

1870 \cs_new_protected:Npn \@@_exec_code_before:
1871 {

```

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...

```

1872 \clist_map_inline:Nn \l_@@_corners_cells_clist
1873 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1874 \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`.

```

1875 \@@_add_to_colors_seq:nn { { nocolor } } { }
1876 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1877 \group_begin:

```

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

```

1878 \if_mode_math:
1879 \@@_exec_code_before_i:
1880 \else:
1881 $ % $
1882 \@@_exec_code_before_i:
1883 $ % $
1884 \fi:
1885 \group_end:
1886 }

```

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`).

```

1887 \cs_new_protected:Npn \@@_exec_code_before_i:
1888 {
1889 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1890 { \@@_rescan_for_spanish:N \l_@@_code_before_tl }

```

Here is the `\CodeBefore`. The construction is a bit complicated because `\g_@@_pre_code_before_tl` may begin with keys between square brackets. Moreover, after the analyze of those keys, we sometimes have to decide to do *not* execute the rest of `\g_@@_pre_code_before_tl` (when it is asked for the creation of cell nodes in the `\CodeBefore`). That's why we use a `\q_stop`: it will be used to discard the rest of `\g_@@_pre_code_before_tl`.

```

1891 \exp_last_unbraced:No \@@_CodeBefore_keys:
1892 \g_@@_pre_code_before_tl

```

Now, all the cells which are specified to be colored by instructions in the `\CodeBefore` will actually be colored. It's a two-stages mechanism because we want to draw all the cells with the same color at the same time to absolutely avoid thin white lines in some PDF viewers.

```

1893 \@@_actually_color:
1894 \l_@@_code_before_tl
1895 \q_stop
1896 }

```

```

1897 \keys_define:nn { nicematrix / CodeBefore }
1898 {
1899 create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1900 sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1901 sub-matrix .value_required:n = true ,
1902 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1903 delimiters / color .value_required:n = true ,
1904 unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1905 }
1906 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1907 {
1908 \keys_set:nn { nicematrix / CodeBefore } { #1 }
1909 \@@_CodeBefore:w
1910 }

```

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.

```

1911 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1912 {
1913 \bool_if:NTF \g_@@_aux_found_bool

```

```

1914 {
1915   \@@_pre_code_before:
1916   \legacy_if:nF { measuring@ } { #1 }
1917 }

```

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

```

1918 {
1919   \pgfsys@markposition { \@@_env: - position }
1920   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1921   \pgfpicture
1922   \pgf@relevantforpicturesizefalse
1923   \endpgfpicture

```

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

```

1924   \pgfpicture
1925   \pgfrememberpicturepositiononpagetrue
1926   \endpgfpicture

```

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

```

1927   \pgfpicture
1928   \pgf@relevantforpicturesizefalse
1929   \pgfrememberpicturepositiononpagetrue
1930   \endpgfpicture

```

The following picture corresponds `\@@_actually_color:`

```

1931   \pgfpicture
1932   \pgf@relevantforpicturesizefalse
1933   \endpgfpicture
1934 }
1935 }

```

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.

```

1936 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1937 {
1938   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1939   {
1940     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1941     \pgfcoordinate { \@@_env: - row - ##1 - base }
1942     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1943     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1944     {
1945       \cs_if_exist:cT
1946       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1947       {
1948         \pgfsys@getposition
1949         { \@@_env: - ##1 - #####1 - NW }
1950         \@@_node_position:
1951         \pgfsys@getposition
1952         { \@@_env: - ##1 - #####1 - SE }
1953         \@@_node_position_i:
1954         \@@_pgf_rect_node:nnn
1955         { \@@_env: - ##1 - #####1 }
1956         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1957         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1958       }
1959     }
1960   }
1961   \@@_create_extra_nodes:
1962   \@@_create_aliases_last:
1963 }

```

```

1964 \cs_new_protected:Npn \@@_create_aliases_last:
1965 {
1966   \int_step_inline:nn \c@iRow
1967   {
1968     \pgfnodealias
1969     { \@@_env: - ##1 - last }
1970     { \@@_env: - ##1 - \int_use:N \c@jCol }
1971   }
1972   \int_step_inline:nn \c@jCol
1973   {
1974     \pgfnodealias
1975     { \@@_env: - last - ##1 }
1976     { \@@_env: - \int_use:N \c@iRow - ##1 }
1977   }
1978   \pgfnodealias
1979   { \@@_env: - last - last }
1980   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1981 }

1982 \cs_new_protected:Npn \@@_create_blocks_nodes:
1983 {
1984   \pgfpicture
1985   \pgf@relevantforpicturesizefalse
1986   \pgfrememberpicturepositiononpagetrue
1987   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1988   { \@@_create_one_block_node:nnnnn ##1 }
1989   \endpgfpicture
1990 }

```

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

```

1991 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1992 {
1993   \tl_if_empty:nF { #5 }
1994   {
1995     \@@_qpoint:n { col - #2 }
1996     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1997     \@@_qpoint:n { #1 }
1998     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1999     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
2000     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
2001     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
2002     % \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y % 2026-02-24
2003     \@@_pgf_rect_node:nnnnn
2004     { \@@_env: - #5 }
2005     { \dim_use:N \l_tmpa_dim }
2006     { \dim_use:N \l_tmpb_dim }
2007     { \dim_use:N \l_@@_tmpc_dim }
2008     { \dim_use:N \pgf@y }
2009   }
2010 }

```

10 The environment `{NiceArrayWithDelims}`

```

2011 \NewDocumentEnvironment { NiceArrayWithDelims }
2012 { m m 0 { } m ! 0 { } t \CodeBefore }
2013 {

```

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


```

2014 \@@_provide_pgfsyspdfmark:
2015 \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.

```

2016 \bgroup

2017 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2018 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2019 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
2020 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

2021 \int_gzero:N \g_@@_block_box_int
2022 \dim_gzero:N \g_@@_width_last_col_dim
2023 \dim_gzero:N \g_@@_width_first_col_dim
2024 \bool_gset_false:N \g_@@_row_of_col_done_bool
2025 \str_if_empty:NT \g_@@_name_env_str
2026 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
2027 \bool_if:NTF \l_@@_tabular_bool
2028 \mode_leave_vertical:
2029 \@@_test_if_math_mode:
2030 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
2031 \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.

```

2032 \cs_gset_eq:cN { @@_old_CT@arc@ } \CT@arc@

```

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

```

2033 \cs_if_exist:NT \tikz@library@external@loaded
2034 {
2035     \tikzexternaldisable
2036     \cs_if_exist:NT \ifstandalone
2037     { \tikzset { external / optimize = false } }
2038 }

```

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

```

2039 \int_gincr:N \g_@@_env_int
2040 \bool_if:NF \l_@@_block_auto_columns_width_bool
2041 { \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.

```

2042 \seq_gclear:N \g_@@_blocks_seq
2043 \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`.

```

2044 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
2045 \seq_gclear:N \g_@@_pos_of_xdots_seq
2046 \tl_gclear_new:N \g_@@_code_before_tl
2047 \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.

```

2048 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2049 {

```

⁸e.g. `\color[rgb]{0.5,0.5,0}`

```

2050     \bool_gset_true:N \g_@@_aux_found_bool
2051     \use:c { g_@@ _ \int_use:N \g_@@_env_int _ t1 }
2052   }
2053   { \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.

```

2054     \tl_gclear:N \g_@@_aux_tl
2055     \tl_if_empty:NF \g_@@_code_before_tl
2056     {
2057       \bool_set_true:N \l_@@_code_before_bool
2058       \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
2059     }
2060     \tl_if_empty:NF \g_@@_pre_code_before_tl
2061     { \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.

```

2062     \bool_if:NTF \g_@@_delims_bool
2063     { \keys_set:nn { nicematrix / pNiceArray } }
2064     { \keys_set:nn { nicematrix / NiceArray } }
2065     { #3 , #5 }

2066     \@@_set_CTarc:o \l_@@_rules_color_tl % noqa: w302

```

The argument #6 is the last argument of {NiceArrayWithDelims}. With that argument of type “t \CodeBefore”, we test whether there is the keyword \CodeBefore at the beginning of the body of the environment. If that keyword is present, we have now to extract all the content between that keyword \CodeBefore and the (other) keyword \Body. It’s the job that will do the command \@@_CodeBefore_Body:w. After that job, the command \@@_CodeBefore_Body:w will go on with \@@_pre_array:.

```

2067     \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
2068   }

```

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

```

2069   {
2070     \bool_if:NTF \l_@@_light_syntax_bool
2071     { \use:c { end @@-light-syntax } }
2072     { \use:c { end @@-normal-syntax } }
2073     $ % $
2074     \skip_horizontal:N \l_@@_right_margin_dim
2075     \skip_horizontal:N \l_@@_extra_right_margin_dim
2076     \hbox_set_end:
2077     \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.

```

2078     \bool_if:NT \l_@@_width_used_bool
2079     {
2080       \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2081       { \@@_error_or_warning:n { width-without-X-columns } }
2082     }

```

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

```

2083     \fp_compare:nNnT \g_@@_total_X_weight_fp > \c_zero_fp
2084     \@@_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).

```

2085 \int_compare:nNtT \l_@@_last_row_int > { -2 }
2086 {
2087   \bool_if:NF \l_@@_last_row_without_value_bool
2088   {
2089     \int_compare:nNfT \l_@@_last_row_int = \c@iRow
2090     {
2091       \@@_error:n { Wrong~last~row }
2092       \int_set_eq:NN \l_@@_last_row_int \c@iRow
2093     }
2094   }
2095 }

```

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

```

2096 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2097 \bool_if:NTF \g_@@_last_col_found_bool
2098 { \int_gdecr:N \c@jCol }
2099 {
2100   \int_compare:nNtT \l_@@_last_col_int > { -1 }
2101   { \@@_error:n { last~col~not~used } }
2102 }

```

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

```

2103 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2104 \int_compare:nNtT \l_@@_last_row_int > { -1 }
2105 { \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. 94).

```

2106 \int_if_zero:nT \l_@@_first_col_int
2107 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2108 \bool_if:nTF { ! \g_@@_delims_bool }
2109 {
2110   \str_if_eq:eeTF c \l_@@_baseline_tl
2111   \@@_use_arraybox_with_notes_c:
2112   {
2113     \str_if_eq:eeTF b \l_@@_baseline_tl
2114     \@@_use_arraybox_with_notes_b:
2115     \@@_use_arraybox_with_notes:
2116   }
2117 }

```

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).

```

2118 {
2119   \int_if_zero:nTF \l_@@_first_row_int
2120   {
2121     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2122     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2123   }
2124   { \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

```

2125 \int_compare:nNtTF \l_@@_last_row_int > { -2 }
2126 {
2127   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim

```

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

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

```

2128         \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2129     }
2130     { \dim_zero:N \l_tmpb_dim }
2131 \hbox_set:Nn \l_tmpa_box
2132 {
2133     \m@th
2134     $ % $
2135     \@@_color:o \l_@@_delimiters_color_tl
2136     \exp_after:wN \left \g_@@_left_delim_tl
2137     \vcenter
2138     {

```

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.

```

2139         \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2140     \hbox
2141     {
2142         \bool_if:NTF \l_@@_tabular_bool
2143         { \skip_horizontal:n { - \tabcolsep } }
2144         { \skip_horizontal:n { - \arraycolsep } }
2145         \@@_use_arraybox_with_notes_c:
2146         \bool_if:NTF \l_@@_tabular_bool
2147         { \skip_horizontal:n { - \tabcolsep } }
2148         { \skip_horizontal:n { - \arraycolsep } }
2149     }

```

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

```

2150         \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2151     }
2152     \exp_after:wN \right \g_@@_right_delim_tl
2153     $ % $
2154 }

```

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.

```

2155     \bool_if:NTF \l_@@_delimiters_max_width_bool
2156     { \@@_put_box_in_flow_bis:nn \g_@@_left_delim_tl \g_@@_right_delim_tl }
2157     \@@_put_box_in_flow:
2158 }

```

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. 95).

```

2159     \bool_if:NT \g_@@_last_col_found_bool
2160     { \skip_horizontal:N \g_@@_width_last_col_dim }
2161     \bool_if:NT \l_@@_preamble_bool
2162     {
2163         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2164         { \@@_err_columns_not_used: }
2165     }
2166     \@@_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.

```

2167     \egroup

```

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

```

2168     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2169     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2170     \iow_now:Ne \@mainaux
2171     {
2172         \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2173         \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2174         { \exp_not:o \g_@@_aux_tl }
2175     }

```

```

2176 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2177 \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2178 }

```

This is the end of the environment {NiceArrayWithDelims}.

```

2179 \cs_new_protected:Npn \@_err_columns_not_used:
2180 {
2181   \@@_warning:n { columns~not~used }
2182   \cs_gset:Npn \@_err_columns_not_used: { }
2183 }

```

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 .

```

2184 \cs_new_protected:Npn \@_compute_width_X:
2185 {
2186   \tl_gput_right:Ne \g_@@_aux_tl
2187   {
2188     \bool_set_true:N \l_@@_X_columns_aux_bool
2189     \dim_set:Nn \l_@@_X_columns_dim
2190   }

```

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`

```

2191   \bool_lazy_and:nnTF \g_@@_V_of_X_bool \l_@@_X_columns_aux_bool
2192   { \dim_use:N \l_@@_X_columns_dim }
2193   {
2194     \dim_compare:nNnTF
2195     {
2196       \dim_abs:n
2197       { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2198     }
2199     <
2200     { 0.001 pt }
2201     { \dim_use:N \l_@@_X_columns_dim }
2202     {
2203       \dim_eval:n
2204       {
2205         \l_@@_X_columns_dim
2206         +
2207         \fp_to_dim:n
2208         {
2209           (
2210             \dim_eval:n
2211             { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2212           )
2213           / \fp_use:N \g_@@_total_X_weight_fp
2214         }
2215       }
2216     }
2217   }
2218 }
2219 }
2220 }

```

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`.

```

2221 \cs_new_protected:Npn \@@_transform_preamble:
2222 {
2223   \@@_transform_preamble_i:
2224   \@@_transform_preamble_ii:
2225 }
2226 \cs_new_protected:Npn \@@_transform_preamble_i:
2227 {
2228   \int_gzero:N \c@jCol

```

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

```

2229   \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.

```

2230   \bool_gset_false:N \g_tmpb_bool

```

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

```

2231   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2232   \int_zero:N \l_tmpa_int
2233   \tl_gclear:N \g_@@_array_preamble_tl
2234   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2235   {
2236     \tl_gset:Nn \g_@@_array_preamble_tl
2237     { ! { \skip_horizontal:N \arrayrulewidth } }
2238   }
2239   {
2240     \clist_if_in:NnT \l_@@_vlines_clist 1
2241     {
2242       \tl_gset:Nn \g_@@_array_preamble_tl
2243       { ! { \skip_horizontal:N \arrayrulewidth } }
2244     }
2245   }

```

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

```

2246   \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2247   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2248   \@@_replace_columncolor:
2249   }

2250 \cs_new_protected:Npn \@@_transform_preamble_ii:
2251 {

```

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

```

2252   \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2253   {
2254     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2255     { \bool_gset_true:N \g_@@_delims_bool }
2256   }
2257   { \bool_gset_true:N \g_@@_delims_bool }

```

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

```
2258 \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).

```
2259 \int_if_zero:nTF \l_@@_first_col_int
2260 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2261 {
2262   \bool_if:NF \g_@@_delims_bool
2263   {
2264     \bool_if:NF \l_@@_tabular_bool
2265     {
2266       \clist_if_empty:NT \l_@@_vlines_clist
2267       {
2268         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2269         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2270       }
2271     }
2272   }
2273 }
2274 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2275 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2276 {
2277   \bool_if:NF \g_@@_delims_bool
2278   {
2279     \bool_if:NF \l_@@_tabular_bool
2280     {
2281       \clist_if_empty:NT \l_@@_vlines_clist
2282       {
2283         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2284         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2285       }
2286     }
2287   }
2288 }
```

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.

```
2289 \tag_if_active:F
2290 {
```

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

```
2291 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2292 {
2293   \tl_gput_right:Nn \g_@@_array_preamble_tl
2294   { > { \@@_err_too_many_cols: } 1 }
2295 }
2296 }
2297 }
```

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.

```
2298 \cs_new_protected:Npn \@@_rec_preamble:n #1
2299 {
```

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

```
2300 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2301 { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2302 {
```

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

```
2303 \cs_if_exist:cTF { NC @ find @ #1 }
2304 {
2305 \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2306 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2307 }
2308 {
2309 \str_if_eq:nnTF { #1 } { S }
2310 { \@@_fatal:n { unknown~column~type~S } }
2311 { \@@_fatal:nn { unknown~column~type } { #1 } }
2312 }
2313 }
2314 }
```

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

```
2315 \cs_new_protected:Npn \@@_c: #1
2316 {
2317 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2318 \tl_gclear:N \g_@@_pre_cell_tl
2319 \tl_gput_right:Nn \g_@@_array_preamble_tl
2320 { > \@@_cell_begin: c < \@@_cell_end: }
```

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

```
2321 \int_gincr:N \c@jCol
2322 \@@_rec_preamble_after_col:n
2323 }

2324 \cs_new_protected:Npn \@@_l: #1
2325 {
2326 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2327 \tl_gclear:N \g_@@_pre_cell_tl
2328 \tl_gput_right:Nn \g_@@_array_preamble_tl
2329 {
2330 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2331 l
2332 < \@@_cell_end:
2333 }
2334 \int_gincr:N \c@jCol
2335 \@@_rec_preamble_after_col:n
2336 }

2337 \cs_new_protected:Npn \@@_r: #1
2338 {
2339 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2340 \tl_gclear:N \g_@@_pre_cell_tl
2341 \tl_gput_right:Nn \g_@@_array_preamble_tl
2342 {
2343 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2344 r
2345 < \@@_cell_end:
2346 }
2347 \int_gincr:N \c@jCol
2348 \@@_rec_preamble_after_col:n
2349 }
```

¹¹We do that because it's an easy way to insert the letter at some places in the code that we will add to `\g_@@_array_preamble_tl`.

For ! and @

```

2350 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2351 {
2352   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2353   \@@_rec_preamble:n
2354 }
2355 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2356 \cs_new_protected:cpn { @@ _ | : } #1
2357 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2358   \int_incr:N \l_tmpa_int
2359   \@@_make_preamble_i_i:n
2360 }

2361 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2362 {

```

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

```

2363   \str_if_eq:nnTF { #1 } { | }
2364   { \use:c { @@ _ | : } | }
2365   { \@@_make_preamble_i_ii:nn { } #1 }
2366 }

```

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

```

2367 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2368 {
2369   \str_if_eq:nnTF { #2 } { [ ]
2370   { \@@_make_preamble_i_iii:nw { #1 } [ ]
2371   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2372   }
2373 \cs_new_protected:Npn \@@_make_preamble_i_iii:nw #1 [ #2 ]
2374 { \@@_make_preamble_i_iii:nn { #1 , #2 } }

2375 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2376 {
2377   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2378   \tl_gput_right:Ne \g_@@_array_preamble_tl
2379   {

```

Here, the command \dim_use:N is mandatory.

```

2380   \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_before_dim }
2381   }
2382   \tl_gput_right:Ne \g_@@_rules_tl
2383   {

```

With the keys of nicematrix / rules-after we would write:

```

\@@_draw_vrule:n
{
  multiplicity = \int_use:N \l_tmpa_int ,
  position = \int_eval:n { \c@jCol + 1 } ,
  total-width = \dim_use:N \l_@@_rule_width_before_dim ,
  #2
}

```

We will use a version slightly more efficient:

```

2384 {
2385   \int_compare:nNnT \l_tmpa_int > 1
2386   { \@@_set_multiplicity:n { \int_use:N \l_tmpa_int } }
2387   \int_set:Nn \l_@@_position_int { \int_eval:n { \c@jCol + 1 } }
2388   \dim_set:Nn \l_@@_rule_width_after_dim

```

```

2389         { \dim_use:N \l_@@_rule_width_before_dim }
2390     \@@_draw_vrule:n { #2 }
2391 }

```

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.

```

2392     }
2393     \int_zero:N \l_tmpa_int
2394     \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2395     \@@_rec_preamble:n #1
2396 }

2397 \cs_new_protected:cpn { @@_> : } #1 #2
2398 {
2399     \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2400     \@@_rec_preamble:n
2401 }

2402 \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.

```

2403 \keys_define:nn { nicematrix / p-column }
2404 {
2405     r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2406     r .value_forbidden:n = true ,
2407     c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2408     c .value_forbidden:n = true ,
2409     l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2410     l .value_forbidden:n = true ,
2411     S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2412     S .value_forbidden:n = true ,
2413     p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2414     p .value_forbidden:n = true ,
2415     t .meta:n = p ,
2416     m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2417     m .value_forbidden:n = true ,
2418     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2419     b .value_forbidden:n = true
2420 }

```

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

```

2421 \cs_new_protected:Npn \@@_p: #1
2422 {
2423     \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2424     \@@_make_preamble_ii_i:n
2425 }
2426 \cs_new_eq:NN \@@_b: \@@_p:
2427 \cs_new_eq:NN \@@_m: \@@_p:

2428 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2429 {
2430     \str_if_eq:nnTF { #1 } { [ ]
2431         { \@@_make_preamble_ii_ii:w [ ]
2432             { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2433         }
2434     \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2435     { \@@_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.

```
2436 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2437 {
```

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).

```
2438 \str_set:Nn \l_@@_hpos_col_str { j }
2439 \@@_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`.

```
2440 \setlength { \l_tmpa_dim } { #2 }
2441 \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2442 }
2443 \cs_new_protected:Npn \@@_keys_p_column:n #1
2444 { \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.

```
2445 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2446 {
```

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

```
2447 \use:e
2448 {
2449 \@@_make_preamble_ii_vi:nnnnnnnn
2450 { \str_if_eq:eeTF p \l_@@_vpos_col_str { t } { b } }
2451 { #1 }
2452 {
2453 \cs_set_eq:NN \rotate \@@_rotate_p_col:
```

The parameter `\l_@@_hpos_col_str` (as `\l_@@_vpos_col_str`) exists only during the construction of the preamble. During the composition of the array itself, you will have, in each cell, the parameter `\l_@@_hpos_cell_tl` which will provide the horizontal alignment of the column to which belongs the cell.

```
2454 \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2455 { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2456 {
```

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

```
2457 \def \exp_not:N \l_@@_hpos_cell_tl
2458 { \str_lowercase:f { \l_@@_hpos_col_str } }
2459 }
2460 \IfPackageLoadedTF { ragged2e }
2461 {
2462 \str_case:on \l_@@_hpos_col_str
2463 {
```

The following `\exp_not:N` are mandatory.

```
2464 c { \exp_not:N \Centering }
2465 l { \exp_not:N \RaggedRight }
2466 r { \exp_not:N \RaggedLeft }
2467 }
2468 }
2469 {
2470 \str_case:on \l_@@_hpos_col_str
2471 {
2472 c { \exp_not:N \centering }
2473 l { \exp_not:N \raggedright }
2474 r { \exp_not:N \raggedleft }
```

```

2475     }
2476   }
2477   #3
2478 }
2479 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2480 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2481 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2482 { #2 }
2483 {
2484   \str_case:onF \l_@@_hpos_col_str
2485   {
2486     { j } { c }
2487     { si } { c }
2488   }

```

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

```

2489     { \str_lowercase:f \l_@@_hpos_col_str }
2490   }
2491 }

```

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

```

2492   \int_gincr:N \c@jCol
2493   \@@_rec_preamble_after_col:n
2494 }

```

#1 is the optional argument of `{minipage}` (or `{varwidth}`): `t` or `b`. Indeed, for the columns of type `m`, we use the value `b` here because there is a special post-action in order to center vertically the box (see **#4**).

#2 is the width of the `{minipage}` (or `{varwidth}`), that is to say also the width of the column.

#3 is the coding for the horizontal position of the content of the cell (`\centering`, `\raggedright`, `\raggedleft` or nothing). It's also possible to put in that **#3** some code to fix the value of `\l_@@_hpos_cell_tl` which will be available in each cell of the column.

#4 is an extra-code which contains `\@@_center_cell_box:` (when the column is a `m` column) or nothing (in the other cases).

#5 is a code put just before the `c` (or `r` or `l`: see **#8**).

#6 is a code put just after the `c` (or `r` or `l`: see **#8**).

#7 is the type of environment: `minipage` or `varwidth`.

#8 is the letter `c` or `r` or `l` which is the basic specifier of column which is used *in fine*.

```

2495 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2496 {
2497   \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2498   {
2499     \tl_gput_right:Nn \g_@@_array_preamble_tl
2500     { > \@@_test_if_empty_for_S: }
2501   }
2502   {
2503     \str_if_eq:eeTF { #7 } { varwidth }
2504     {
2505       \tl_gput_right:Nn \g_@@_array_preamble_tl
2506       { > \@@_test_if_empty_varwidth: }
2507     }
2508     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2509   }
2510   \tl_gput_right:Nn \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2511   \tl_gclear:N \g_@@_pre_cell_tl
2512   \tl_gput_right:Nn \g_@@_array_preamble_tl
2513   {
2514     > {

```

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.

```

2515     \dim_set:Nn \l_@@_col_width_dim { #2 }
2516     \@@_cell_begin:

```

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

```
2517 \use:c { #7 } [ #1 ] { #2 }
```

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

```
2518 \everypar
2519 {
2520 \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2521 \everypar { }
2522 }
```

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

```
2523 #3
```

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

```
2524 \g_@@_row_style_tl
2525 \arraybackslash
2526 #5
2527 }
2528 #8
2529 < {
2530 #6
```

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

```
2531 \@finalstrut \@arstrutbox
2532 \use:c { end #7 }
```

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

```
2533 #4
2534 \@@_cell_end:
2535 }
2536 }
2537 }
```

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

```
2538 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2539 {
```

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 `&`).

```
2540 \group_align_safe_begin:
2541 \peek_meaning:NTF &
2542 \@@_the_cell_is_empty:
2543 {
2544 \peek_meaning:NTF \\\
2545 \@@_the_cell_is_empty:
2546 {
2547 \peek_meaning:NTF \crrc
2548 \@@_the_cell_is_empty:
2549 \group_align_safe_end:
2550 }
2551 }
2552 }
```

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

```
2553 \cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
2554 {
2555 \group_align_safe_begin:
2556 \peek_meaning:NTF &
2557 \@@_the_cell_is_empty_varwidth:
2558 {
```

```

2559     \peek_meaning:NTF \\\
2560     \@@_the_cell_is_empty_varwidth:
2561     {
2562         \peek_meaning:NTF \crcr
2563         \@@_the_cell_is_empty_varwidth:
2564         \group_align_safe_end:
2565     }
2566 }
2567 }

2568 \cs_new_protected:Npn \@@_the_cell_is_empty:
2569 {
2570     \group_align_safe_end:
2571     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2572     {

```

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

```

2573     \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).

```

2574     \skip_horizontal:N \l_@@_col_width_dim
2575 }
2576 }

2577 \cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
2578 {
2579     \group_align_safe_end:
2580     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2581     { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2582 }

2583 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2584 {
2585     \peek_meaning:NT \__siunitx_table_skip:n
2586     { \bool_gset_true:N \g_@@_empty_cell_bool }
2587 }

```

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

```

2588 \cs_new_protected:Npn \@@_center_cell_box:
2589 {

```

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

```

2590     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2591     {
2592         \dim_compare:nNnT
2593         { \box_ht:N \l_@@_cell_box }
2594         >

```

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).

```

2595         { \box_ht:N \strutbox }
2596     {
2597         \hbox_set:Nn \l_@@_cell_box
2598         {
2599             \box_move_down:nn
2600             {
2601                 ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox

```

```

2602             + \baselineskip ) / 2
2603         }
2604         { \box_use:N \l_@@_cell_box }
2605     }
2606 }
2607 }
2608 }

```

For V (similar to the V of varwidth).

```

2609 \cs_new_protected:Npn \@@_V: #1 #2
2610 {
2611     \str_if_eq:nnTF { #2 } { [ ]
2612         { \@@_make_preamble_V_i:w [ ]
2613           { \@@_make_preamble_V_i:w [ ] { #2 } }
2614         }
2615     \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2616     { \@@_make_preamble_V_ii:nn { #1 } }
2617     \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2618     {
2619         \str_set:Nn \l_@@_vpos_col_str { p }
2620         \str_set:Nn \l_@@_hpos_col_str { j }
2621         \@@_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`.

```

2622     \setlength { \l_tmpa_dim } { #2 }
2623     \IfPackageLoadedTF { varwidth }
2624     { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2625     {
2626         \@@_error_or_warning:n { varwidth~not~loaded }
2627         \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2628     }
2629 }
2630 % \end{macrocod}
2631 %
2632 % \medskip
2633 % For |w| and |W|
2634 % \begin{macrocode}
2635 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2636 \cs_new_protected:Npn \@@_W: { \@@_make_preamble_w:nnnn { \@@_special_W: } }

```

#1 is a special argument: empty for `w` and equal to `\@@_special_W:` for `W`;

#2 is the type of column (`w` or `W`);

#3 is the type of horizontal alignment (`c`, `l`, `r` or `s`);

#4 is the width of the column. It is provided by curryfication.

```

2637 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3
2638 {
2639     \tl_if_in:nnTF { clr } { #3 }
2640     { \@@_make_preamble_w_i:nnnn { #1 } { #2 } { #3 } }
2641     {
2642         \@@_error:nn { Invalid~argument~for~w } { #3 }
2643         \@@_make_preamble_w_i:nnnn { #1 } { #2 } { c }
2644     }
2645 }
2646 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2 #3 #4
2647 {
2648     \str_if_eq:nnTF { #3 } { s }
2649     { \@@_make_preamble_w_ii:nnnn { #1 } { #4 } }
2650     { \@@_make_preamble_w_iii:nnnn { #1 } { #2 } { #3 } { #4 } }
2651 }

```

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.

```

2652 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2
2653 {
2654   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2655   \tl_gclear:N \g_@@_pre_cell_tl
2656   \tl_gput_right:Nn \g_@@_array_preamble_tl
2657   {
2658     > {

```

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`.

```

2659       \setlength { \l_@@_col_width_dim } { #2 }
2660       \@@_cell_begin:
2661       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2662     }
2663     c
2664     < {
2665       \@@_cell_end_for_w_s:
2666       #1
2667       \@@_adjust_size_box:
2668       \box_use_drop:N \l_@@_cell_box
2669     }
2670   }
2671   \int_gincr:N \c@jCol
2672   \@@_rec_preamble_after_col:n
2673 }

```

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

```

2674 \cs_new_protected:Npn \@@_make_preamble_w_iii:nnnn #1 #2 #3 #4
2675 {
2676   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2677   \tl_gclear:N \g_@@_pre_cell_tl
2678   \tl_gput_right:Nn \g_@@_array_preamble_tl
2679   {
2680     > {

```

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`.

```

2681       \setlength { \l_@@_col_width_dim } { #4 }
2682       \hbox_set:Nw \l_@@_cell_box
2683       \@@_cell_begin:
2684       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2685     }
2686     c
2687     < {
2688       \@@_cell_end:
2689       \hbox_set_end:
2690       #1
2691       \@@_adjust_size_box:
2692       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2693     }
2694   }

```

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

```

2695   \int_gincr:N \c@jCol
2696   \@@_rec_preamble_after_col:n
2697 }

```



```

2698 \cs_new_protected:Npn \@@_special_W:
2699 {
2700   \dim_compare:nNtF { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2701   { \@@_warning:n { W~warning } }
2702 }

```

For S (of siunitx).

```

2703 \cs_new_protected:Npn \@@_S: #1 #2
2704 {
2705   \str_if_eq:nnTF { #2 } { [ ] }
2706   { \@@_make_preamble_S:w [ ] }
2707   { \@@_make_preamble_S:w [ ] { #2 } }
2708 }
2709 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2710 { \@@_make_preamble_S_i:n { #1 } }
2711 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2712 {
2713   \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2714   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2715   \tl_gclear:N \g_@@_pre_cell_tl
2716   \tl_gput_right:Nn \g_@@_array_preamble_tl
2717   {
2718     > {

```

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).

```

2719       \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2720       \keys_set:nn { siunitx } { #1 }
2721       \@@_cell_begin:
2722       \siunitx_cell_begin:w
2723     }
2724     c
2725     <
2726     {
2727       \siunitx_cell_end:

```

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

```

2728       \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2729       {
2730         \bool_if:NTF \l__siunitx_table_text_bool
2731         \bool_set_true:N
2732         \bool_set_false:N
2733         \l__siunitx_table_text_bool
2734       }
2735       \@@_cell_end:
2736     }
2737   }

```

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

```

2738     \int_gincr:N \c@jCol
2739     \@@_rec_preamble_after_col:n
2740   }

```

For (, [and \{.

```

2741 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2742 {
2743   \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.

```

2744 \int_if_zero:nTF \c@jCol
2745 {
2746   \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2747   {

```

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

```

2748     \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2749     \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2750     \@@_rec_preamble:n #2
2751   }
2752   {
2753     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2754     \@@_make_preamble_iv:nn { #1 } { #2 }
2755   }
2756 }
2757 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2758 }
2759 \cs_set_eq:cc { @@ _ \token_to_str:N [ : ] { @@ _ \token_to_str:N ( : }
2760 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2761 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2762 {
2763   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2764   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2765   \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2766   {
2767     \@@_error:nn { delimiter~after~opening } { #2 }
2768     \@@_rec_preamble:n
2769   }
2770   { \@@_rec_preamble:n #2 }
2771 }

```

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

```

2772 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2773 { \use:c { @@ _ \token_to_str:N ( : } }

```

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

```

2774 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2775 {
2776   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2777   \tl_if_in:nnTF { ) ] \} } { #2 }
2778   { \@@_make_preamble_v:nnn #1 #2 }
2779   {
2780     \str_if_eq:nnTF \s_stop { #2 }
2781     {
2782       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2783       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2784       {
2785         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2786         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2787         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2788         \@@_rec_preamble:n #2
2789       }
2790     }
2791     {
2792       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2793       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2794       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2795       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }

```

```

2796         \@@_rec_preamble:n #2
2797     }
2798 }
2799 }
2800 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2801 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2802 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2803 {
2804     \str_if_eq:nnTF \s_stop { #3 }
2805     {
2806         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2807         {
2808             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2809             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2810             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2811             \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2812         }
2813         {
2814             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2815             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2816             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2817             \@@_error:nn { double~closing~delimiter } { #2 }
2818         }
2819     }
2820     {
2821         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2822         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2823         \@@_error:nn { double~closing~delimiter } { #2 }
2824         \@@_rec_preamble:n #3
2825     }
2826 }
2827 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2828 { \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 @{...}.

```

2829 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2830 {
2831     \str_if_eq:nnTF { #1 } { < }
2832     { \@@_rec_preamble_after_col_i:n }
2833     {
2834         \str_if_eq:nnTF { #1 } { @ }
2835         { \@@_rec_preamble_after_col_ii:n }
2836         {
2837             \str_if_eq:eeTF \l_@@_vlines_clist { all }
2838             {
2839                 \tl_gput_right:Nn \g_@@_array_preamble_tl
2840                 { ! { \skip_horizontal:N \arrayrulewidth } }
2841             }
2842             {
2843                 \clist_if_in:NeT \l_@@_vlines_clist
2844                 { \int_eval:n { \c@jCol + 1 } }
2845                 {
2846                     \tl_gput_right:Nn \g_@@_array_preamble_tl
2847                     { ! { \skip_horizontal:N \arrayrulewidth } }
2848                 }
2849             }
2850             \@@_rec_preamble:n { #1 }
2851         }
2852     }
2853 }

```

```

2854 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2855 {
2856   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2857   \@@_rec_preamble_after_col:n
2858 }

```

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.

```

2859 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2860 {
2861   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2862   {
2863     \tl_gput_right:Nn \g_@@_array_preamble_tl
2864     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2865   }
2866   {
2867     \clist_if_in:NcTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2868     {
2869       \tl_gput_right:Nn \g_@@_array_preamble_tl
2870       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2871     }
2872     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2873   }
2874   \@@_rec_preamble:n
2875 }

```

```

2876 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2877 {
2878   \tl_clear:N \l_tmpa_tl
2879   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2880   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2881 }

```

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

```

2882 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2883 { \@@_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.

```

2884 \cs_new_protected:Npn \@@_X: #1 #2
2885 {
2886   \str_if_eq:nnTF { #2 } { [ ]
2887     { \@@_make_preamble_X:w [ ] }
2888     { \@@_make_preamble_X:w [ ] #2 }
2889   }
2890   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2891   { \@@_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`.

```

2892 \keys_define:nn { nicematrix / X-column }
2893 {
2894   V .code:n =
2895   \IfPackageLoadedTF { varwidth }
2896   {
2897     \bool_set_true:N \l_@@_V_of_X_bool

```

```

2898     \bool_gset_true:N \g_@@_V_of_X_bool
2899   }
2900   { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
2901   unknown .code:n =
2902     \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2903     { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2904     { \@@_error_or_warning:n { invalid~weight } }
2905   }

```

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

```

2906 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2907 {

```

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).

```

2908   \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).

```

2909   \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`).

```

2910   \fp_set:Nn \l_tmpa_fp { 1.0 }
2911   \@@_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.

```

2912   \bool_set_false:N \l_@@_V_of_X_bool
2913   \keys_set:no { nicematrix / X-column } \l_tmpa_tl

```

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

```

2914   \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).

```

2915   \bool_if:NTF \l_@@_X_columns_aux_bool
2916   {
2917     \@@_make_preamble_ii_iv:nnn

```

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

```

2918     { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2919     { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
2920     { \@@_no_update_width: }
2921   }

```

In the current compilation, we don't know the actual width of the X column. However, you have to construct the cells of that column! By convention, we have decided to compose in a `{minipage}` of width 5 cm even though we will nullify `\l_@@_cell_box` after its composition.

```

2922   {
2923     \tl_gput_right:Nn \g_@@_array_preamble_tl
2924     {
2925       > {
2926         \@@_cell_begin:
2927         \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`.

```

2928     \NotEmpty

```

The following code will nullify the box of the cell.

```

2929     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2930     { \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`.

```

2931         \begin { minipage } { 5 cm } \arraybackslash
2932     }
2933     c
2934     < {
2935         \end { minipage }
2936         \@@_cell_end:
2937     }
2938 }
2939 \int_gincr:N \c@jCol
2940 \@@_rec_preamble_after_col:n
2941 }
2942 }

2943 \cs_new_protected:Npn \@@_no_update_width:
2944 {
2945     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2946     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2947 }

```

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

```

2948 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2949 {
2950     \seq_gput_right:Ne \g_@@_cols_vlism_seq
2951     { \int_eval:n { \c@jCol + 1 } }
2952     \tl_gput_right:Ne \g_@@_array_preamble_tl
2953     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2954     \@@_rec_preamble:n
2955 }

```

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.

```

2956 \cs_set_eq:cN { @@ _ \token_to_str:N \s_stop : } \use_none:n

```

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

```

2957 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2958 { \@@_fatal:n { Preamble-forgotten } }
2959 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
2960 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
2961 { @@ _ \token_to_str:N \hline : }
2962 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
2963 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
2964 { @@ _ \token_to_str:N \hline : }
2965 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
2966 { @@ _ \token_to_str:N \hline : }
2967 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
2968 { @@ _ \token_to_str:N \hline : }
2969 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
2970 { @@ _ \token_to_str:N \hline : }
2971 \cs_new_protected:cpn { @@ _ \token_to_str:N \linewidth : }
2972 { \@@_fatal:n { NiceTabularX~probably~required } }
2973 \cs_set_eq:cc { @@ _ \token_to_str:N \textwidth : }
2974 { @@ _ \token_to_str:N \linewidth : }

```

12 The redefinition of `\multicolumn`

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

```
2975 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2976 {
```

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`.

```
2977 \multispan { #1 }
2978 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2979 \begingroup
2980 \tbl_update_multicolumn_cell_data:n { #1 }
```

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

```
2981 \tl_gclear:N \g_@@_preamble_tl
2982 \@@_make_m_preamble:n #2 \q_stop
```

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

```
2983 \def \@addamp
2984 {
2985   \legacy_if:nTF { @firstamp }
2986   { \legacy_if_set_false:n { @firstamp } }
2987   { \@preamerr 5 }
2988 }
2989 \exp_args:No \@mkpream \g_@@_preamble_tl
2990 \@addtopreamble \@empty
2991 \endgroup
2992 \UseTaggingSocket { tbl / colspan } { #1 }
```

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

```
2993 \int_compare:nNtT { #1 } > 1
2994 {
2995   \seq_gput_right:Ne \g_@@_multicolumn_cells_seq          % left replaced by right
2996   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2997   \seq_gput_right:Nn \g_@@_multicolumn_sizes_seq { #1 } % left replaced by right
2998   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2999   {
3000     {
3001       \int_if_zero:nTF \c@jCol
3002       { \int_eval:n { \c@iRow + 1 } }
3003       { \int_use:N \c@iRow }
3004     }
3005     { \int_eval:n { \c@jCol + 1 } }
3006     {
3007       \int_if_zero:nTF \c@jCol
3008       { \int_eval:n { \c@iRow + 1 } }
3009       { \int_use:N \c@iRow }
3010     }
3011     { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block.

```
3012   { }
3013 }
3014 }
```

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

```
3015 \RenewDocumentCommand { \cellcolor } { 0 { } m }
3016 {
```

```

3017     \tl_gput_right:Nn \g_@@_pre_code_before_tl
3018     {
3019         \@@_rectanglecolor [ ##1 ]
3020         { \exp_not:n { ##2 } }
3021         { \int_use:N \c@iRow - \int_use:N \c@jCol }
3022         { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
3023     }
3024     \ignorespaces
3025 }

```

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

```

3026     \def \@sharp { #3 }
3027     \@arstrut
3028     \@preamble
3029     \null

```

We add some lines.

```

3030     \int_gadd:Nn \c@jCol { #1 - 1 }
3031     \int_compare:nNnT \c@jCol > \g_@@_col_total_int
3032     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3033     \ignorespaces
3034 }

```

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`.

```

3035 \cs_new_protected:Npn \@@_make_m_preamble:n #1
3036 {
3037     \str_case:nnF { #1 }
3038     {
3039         c { \@@_make_m_preamble_i:n #1 }
3040         l { \@@_make_m_preamble_i:n #1 }
3041         r { \@@_make_m_preamble_i:n #1 }
3042         > { \@@_make_m_preamble_ii:nn #1 }
3043         ! { \@@_make_m_preamble_ii:nn #1 }
3044         @ { \@@_make_m_preamble_ii:nn #1 }
3045         | { \@@_make_m_preamble_iii:n #1 }
3046         p { \@@_make_m_preamble_iv:nnn t #1 }
3047         m { \@@_make_m_preamble_iv:nnn c #1 }
3048         b { \@@_make_m_preamble_iv:nnn b #1 }
3049         w { \@@_make_m_preamble_v:nnnn { } #1 }
3050         W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
3051         \q_stop { }
3052     }
3053     {
3054         \cs_if_exist:cTF { NC @ find @ #1 }
3055         {
3056             \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
3057             \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
3058         }
3059         {
3060             \str_if_eq:nnTF { #1 } { S }
3061             { \@@_fatal:n { unknown~column~type~S~multicolumn } }
3062             { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
3063         }
3064     }
3065 }

```

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

```

3066 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
3067 {
3068     \tl_gput_right:Nn \g_@@_preamble_tl
3069     {

```



```

3070      > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
3071      #1
3072      < \@@_cell_end:
3073    }

```

We test for the presence of a <.

```

3074      \@@_make_m_preamble_x:n
3075    }

```

For >, ! and @

```

3076 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
3077 {
3078   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
3079   \@@_make_m_preamble:n
3080 }

```

For |

```

3081 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
3082 {
3083   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
3084   \@@_make_m_preamble:n
3085 }

```

For p, m and b

```

3086 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
3087 {
3088   \tl_gput_right:Nn \g_@@_preamble_tl
3089   {
3090     > {
3091       \@@_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`.

```

3092       \setlength { \l_tmpa_dim } { #3 }
3093       \begin { minipage } [ #1 ] { \l_tmpa_dim }
3094       \mode_leave_vertical:
3095       \arraybackslash
3096       \vrule height \box_ht:N \@arstrutbox depth \c_zero_dim width \c_zero_dim
3097     }
3098     c
3099     < {
3100       \vrule height \c_zero_dim depth \box_dp:N \@arstrutbox width \c_zero_dim
3101       \end { minipage }
3102       \@@_cell_end:
3103     }
3104   }

```

We test for the presence of a <.

```

3105      \@@_make_m_preamble_x:n
3106    }

```

For w and W

```

3107 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3108 {
3109   \tl_gput_right:Nn \g_@@_preamble_tl
3110   {
3111     > {
3112       \dim_set:Nn \l_@@_col_width_dim { #4 }
3113       \hbox_set:Nw \l_@@_cell_box
3114       \@@_cell_begin:
3115       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3116     }
3117     c

```

```

3118 < {
3119     \@@_cell_end:
3120     \hbox_set_end:
3121     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3122     #1
3123     \@@_adjust_size_box:
3124     \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3125 }
3126 }

```

We test for the presence of a <.

```

3127 \@@_make_m_preamble_x:n
3128 }

```

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

```

3129 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3130 {
3131     \str_if_eq:nnTF { #1 } { < }
3132     \@@_make_m_preamble_ix:n
3133     { \@@_make_m_preamble:n { #1 } }
3134 }
3135 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3136 {
3137     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3138     \@@_make_m_preamble_x:n
3139 }

```

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).

```

3140 \cs_new_protected:Npn \@@_put_box_in_flow:
3141 {
3142     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3143     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3144     \str_if_eq:eeTF \l_@@_baseline_tl { c }
3145     { \box_use_drop:N \l_tmpa_box }
3146     \@@_put_box_in_flow_i:
3147 }

```

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

```

3148 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3149 {
3150     \pgfpicture
3151     \@@_qpoint:n { row - 1 }
3152     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3153     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3154     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3155     \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).

```

3156     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3157     {
3158         \int_set:Nn \l_tmpa_int
3159         { \str_range:Nnn \l_@@_baseline_tl { 6 } { -1 } }
3160         \bool_lazy_or:nnT
3161         { \int_compare_p:nNn \l_tmpa_int < { 1 } }
3162         { \int_compare_p:nNn \l_tmpa_int > { \c@iRow + 1 } }
3163         {
3164             \@@_error:n { bad-value-for-baseline-line }

```

```

3165         \int_set:Nn \l_tmpa_int 1
3166     }
3167     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3168 }
3169 {
3170     \str_if_eq:eeTF t \l_@@_baseline_tl
3171     { \int_set:Nn \l_tmpa_int 1 }
3172     {
3173         \str_if_eq:eeTF b \l_@@_baseline_tl
3174         { \int_set_eq:NN \l_tmpa_int \c@iRow }
3175         { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3176     }
3177     \bool_lazy_or:nnT
3178     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3179     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3180     {
3181         \@@_error:n { bad-value-for-baseline }
3182         \int_set:Nn \l_tmpa_int 1
3183     }
3184     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3185         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3186     }
3187     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3188     \endpgfpicture
3189     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3190     \box_use_drop:N \l_tmpa_box
3191 }

```

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).

```

3192 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3193 {

```

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.

```

3194     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3195     {
3196         \int_compare:nNnT \c@jCol > 1
3197         {
3198             \box_set_wd:Nn \l_@@_the_array_box
3199             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3200         }
3201     }

```

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

```

3202     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3203     \bool_if:NT \l_@@_caption_above_bool
3204     {
3205         \tl_if_empty:NF \l_@@_caption_tl
3206         {
3207             \bool_set_false:N \g_@@_caption_finished_bool
3208             \int_gzero:N \c@tabularnote
3209             \@@_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).

```

3210         \int_compare:nNt \g_@@_notes_caption_int > \c_zero_int
3211         {
3212             \tl_gput_right:Ne \g_@@_aux_tl
3213             {
3214                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3215                 { \int_use:N \g_@@_notes_caption_int }
3216             }
3217             \int_gzero:N \g_@@_notes_caption_int
3218         }
3219     }
3220 }

```

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

```

3221 \hbox
3222 {
3223     \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.

```

3224     \@@_create_extra_nodes:
3225     \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3226 }

```

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).

```

3227 \bool_lazy_any:nT
3228 {
3229     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3230     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3231     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3232 }
3233 {
3234     \bool_if:NTF \l_@@_notes_no_print_bool
3235     { \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes: }
3236     \@@_tabular_notes:
3237 }
3238 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3239 \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3240 \end { minipage }
3241 }

```

```

3242 \cs_new_protected:Npn \@@_insert_caption:
3243 {
3244     \tl_if_empty:NF \l_@@_caption_tl
3245     {
3246         \cs_if_exist:NTF \c@type
3247         { \@@_insert_caption_i: }
3248         { \@@_error:n { caption~outside~float } }
3249     }
3250 }

```

```

3251 \cs_new_protected:Npn \@@_insert_caption_i:
3252 {
3253     \group_begin:

```

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

```

3254     \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.

```

3255 \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
3256 \tl_if_empty:NTF \l_@@_short_caption_tl
3257 \caption
3258 { \caption [ \l_@@_short_caption_tl ] }
3259 { \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.

```

3260 \bool_if:NF \g_@@_caption_finished_bool
3261 {
3262   \bool_gset_true:N \g_@@_caption_finished_bool
3263   \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3264   \int_gzero:N \c@tabularnote
3265 }
3266 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3267 \group_end:
3268 }
3269 \cs_new_protected:Npn \@_tabularnote_error:n #1
3270 {
3271   \@_error_or_warning:n { tabularnote-below-the-tabular }
3272   \cs_gset:Npn \@_tabularnote_error:n ##1 { }
3273 }
3274 \cs_set_protected:Npn \@_tabular_notes_error:
3275 { \@_error:n { Bad-use-of-NiceTabularNotes } }
3276 \cs_set_eq:NN \NiceTabularNotes \@_tabular_notes_error:
3277 \cs_set_protected:Npn \@_tabular_notes:
3278 {
3279   \cs_gset_eq:NN \NiceTabularNotes \@_tabular_notes_error:
3280   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3281   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3282   \skip_vertical:N 0.65ex

```

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

```

3283 \group_begin:
3284 \l_@@_notes_code_before_tl
3285 \tl_if_empty:NF \g_@@_tabularnote_tl
3286 {
3287   \g_@@_tabularnote_tl \par
3288   \tl_gclear:N \g_@@_tabularnote_tl
3289 }

```

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.

```

3290 \int_compare:nNnT \c@tabularnote > \c_zero_int
3291 {
3292   \bool_if:NTF \l_@@_notes_para_bool
3293   {
3294     \begin { tabularnotes* }
3295     \seq_map_inline:Nn \g_@@_notes_seq
3296     { \@_one_tabularnote:nm ##1 }
3297     \strut
3298     \end { tabularnotes* }

```

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

```

3299 \par

```

```

3300     }
3301     {
3302         \tabularnotes
3303         \seq_map_inline:Nn \g_@@_notes_seq
3304         { \@@_one_tabularnote:nn ##1 }
3305         \strut
3306         \endtabularnotes
3307     }
3308 }
3309 \unskip
3310 \group_end:
3311 \bool_if:NT \l_@@_notes_bottomrule_bool
3312 {
3313     \IfPackageLoadedTF { booktabs }
3314     {

```

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

```

3315         \skip_vertical:N \aboverulesep

```

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

```

3316         { \CT@arc@ \hrule height \heavyrulewidth }
3317     }
3318     { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3319 }
3320 \l_@@_notes_code_after_tl
3321 \seq_gclear:N \g_@@_notes_seq
3322 \seq_gclear:N \g_@@_notes_in_caption_seq
3323 \int_gzero:N \c@tabularnote
3324 }

```

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

```

3325 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3326 {
3327     \tl_if_novalue:nTF { #1 }
3328     { \item }
3329     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3330 }

```

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`.

```

3331 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3332 {
3333     \pgfpicture
3334     \@@_qpoint:n { row - 1 }
3335     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3336     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3337     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3338     \endpgfpicture
3339     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3340     \int_if_zero:nT \l_@@_first_row_int
3341     {
3342         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3343         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3344     }
3345     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3346 }

```

Now, the general case.

```

3347 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3348 {

```

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

```

3349 \str_if_eq:eeT \l_@@_baseline_tl { t }
3350 { \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`.

```

3351 \pgfpicture
3352 \@@_qpoint:n { row - 1 }
3353 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3354 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3355 {
3356   \int_set:Nn \l_tmpa_int
3357   {
3358     \str_range:Nnn
3359     \l_@@_baseline_tl
3360     { 6 }
3361     { \tl_count:o \l_@@_baseline_tl }
3362   }
3363   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3364 }
3365 {
3366   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3367   \bool_lazy_or:nnT
3368   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3369   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3370   {
3371     \@@_error:n { bad-value-for~baseline }
3372     \int_set:Nn \l_tmpa_int 1
3373   }
3374   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3375 }
3376 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3377 \endpgfpicture
3378 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3379 \int_if_zero:nT \l_@@_first_row_int
3380 {
3381   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3382   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3383 }
3384 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3385 }

```

The command `\@@_put_box_in_flow_bis:` is used when the option `delimiters/max-width` is used because, in this case, we have to adjust the widths of the delimiters. The arguments `#1` and `#2` are the delimiters specified by the user.

```

3386 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3387 {

```

We will compute the real width of both delimiters used.

```

3388 \dim_zero_new:N \l_@@_real_left_delim_dim
3389 \dim_zero_new:N \l_@@_real_right_delim_dim
3390 \hbox_set:Nn \l_tmpb_box
3391 {
3392   \m@th
3393   $ % $
3394   \left #1
3395   \vcenter
3396   {
3397     \vbox_to_ht:nn
3398     { \box_ht_plus_dp:N \l_tmpa_box }
3399     { }
3400   }
3401   \right .

```

```

3402     $ % $
3403   }
3404   \dim_set:Nn \l_@@_real_left_delim_dim
3405     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3406   \hbox_set:Nn \l_tmpb_box
3407     {
3408       \m@th
3409       $ % $
3410       \left .
3411       \vbox_to_ht:nn
3412         { \box_ht_plus_dp:N \l_tmpa_box }
3413         { }
3414       \right #2
3415       $ % $
3416     }
3417   \dim_set:Nn \l_@@_real_right_delim_dim
3418     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3419   \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3420   \@@_put_box_in_flow:
3421   \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3422 }

```

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).

```

3423 \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).

```

3424 {
3425   \peek_remove_spaces:n
3426   {
3427     \peek_meaning:NTF \end
3428     \@@_analyze_end:Nn
3429     {
3430       \@@_transform_preamble:
3431       \@@_array:o \g_@@_array_preamble_tl
3432     }
3433   }
3434 }
3435 {
3436   \@@_create_col_nodes:
3437   \endarray
3438 }

```

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

```

3439 \NewDocumentEnvironment { @@-light-syntax } { b }
3440 {

```

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.

```

3441   \tl_if_empty:nT { #1 }
3442     { \@@_fatal:n { empty-environment } }
3443   \tl_if_in:nnT { #1 } { & }
3444     { \@@_fatal:n { ampersand-in~light-syntax } }

```



```

3445 \tl_if_in:nnT { #1 } { \ }
3446 { \@@_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`.

```

3447 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3448 }

```

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.

```

3449 {
3450   \@@_create_col_nodes:
3451   \endarray
3452 }

3453 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3454 {
3455   \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).

```

3456 \seq_clear_new:N \l_@@_rows_seq

```

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

```

3457 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3458 \bool_if:NTF \l_@@_light_syntax_expanded_bool
3459   \seq_set_split:Nee
3460   \seq_set_split:Nnon
3461   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3462 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3463 \tl_if_empty:NF \l_tmpa_tl
3464 { \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.

```

3465 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3466 { \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`).

```

3467 \tl_build_begin:N \l_@@_new_body_tl
3468 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3469 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3470 \@@_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).

```

3471 \seq_map_inline:Nn \l_@@_rows_seq
3472 {
3473   \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3474   \@@_line_with_light_syntax:n { ##1 }
3475 }
3476 \tl_build_end:N \l_@@_new_body_tl

```

```

3477 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3478 {
3479     \int_set:Nn \l_@@_last_col_int
3480     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3481 }

```

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.

```

3482 \@@_transform_preamble:

3483 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3484 }

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

```

3497 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3498 {
3499     \str_if_eq:eeT \g_@@_name_env_str { #2 }
3500     { \@@_fatal:n { empty~environment } }

```

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

```

3501 \end { #2 }
3502 }

```

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`).

```

3503 \cs_new:Npn \@@_create_col_nodes:
3504 {
3505     \crrc
3506     \int_if_zero:nT \l_@@_first_col_int
3507     {
3508         \omit
3509         \hbox_overlap_left:n
3510         {
3511             \bool_if:NT \l_@@_code_before_bool
3512             { \pgfsys@markposition { \@@_env: - col - 0 } }
3513             \pgfpicture
3514             \pgfrememberpicturepositiononpagetrue
3515             \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3516             \str_if_empty:NF \l_@@_name_str
3517             { \pgfnodelalias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3518             \endpgfpicture
3519             \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3520         }
3521         &
3522     }
3523     \omit

```

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

```
3524 \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`).

```
3525 \int_if_zero:nTF \l_@@_first_col_int
3526 {
3527   \@@_mark_position:n { 1 }
3528   \pgfpicture
3529   \pgfrememberpicturepositiononpagetrue
3530   \pgfcoordinate { \@@_env: - col - 1 }
3531   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3532   \str_if_empty:NF \l_@@_name_str
3533   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3534   \endpgfpicture
3535 }
3536 {
3537   \bool_if:NT \l_@@_code_before_bool
3538   {
3539     \hbox
3540     {
3541       \skip_horizontal:n { 0.5 \arrayrulewidth }
3542       \pgfsys@markposition { \@@_env: - col - 1 }
3543       \skip_horizontal:n { -0.5 \arrayrulewidth }
3544     }
3545   }
3546   \pgfpicture
3547   \pgfrememberpicturepositiononpagetrue
3548   \pgfcoordinate { \@@_env: - col - 1 }
3549   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3550   \@@_node_alias:n { 1 }
3551   \endpgfpicture
3552 }
```

We compute in `\g_tmpa_skip` the common width of the columns (it's a skip and not a dimension). We use a global variable because we are in a cell of an `\halign` and because we have to use that variable in other cells (of the same row). The affectation of `\g_tmpa_skip`, like all the affectations, must be done after the `\omit` of the cell.

We give a default value for `\g_tmpa_skip` (0 pt plus 1 fill) but we will add some dimensions to it.

```
3553 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3554 \bool_if:NF \l_@@_auto_columns_width_bool
3555 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3556 {
3557   \bool_lazy_and:nnTF
3558   \l_@@_auto_columns_width_bool
3559   { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3560   { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3561   { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3562   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3563 }
3564 \skip_horizontal:N \g_tmpa_skip
3565 \hbox
3566 {
3567   \@@_mark_position:n { 2 }
3568   \pgfpicture
3569   \pgfrememberpicturepositiononpagetrue
3570   \pgfcoordinate { \@@_env: - col - 2 }
3571   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3572   \@@_node_alias:n { 2 }
3573   \endpgfpicture
3574 }
```

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.

```

3575 \int_gset:Nn \g_tmpa_int 1
3576 \bool_if:NTF \g_@@_last_col_found_bool
3577 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3578 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3579 {
3580   &
3581   \omit
3582   \int_gincr:N \g_tmpa_int

```

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

```

3583 \skip_horizontal:N \g_tmpa_skip
3584 \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

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

```

3585 \pgfpicture
3586 \pgfrememberpicturepositiononpagetrue
3587 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3588 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3589 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3590 \endpgfpicture
3591 }

```

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).

```

3592 \bool_lazy_or:nnF
3593 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3594 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3595 {
3596   &
3597   \omit
3598   \skip_horizontal:N \g_tmpa_skip
3599   \int_gincr:N \g_tmpa_int
3600   \bool_lazy_any:nF
3601   {
3602     \g_@@_delims_bool
3603     \l_@@_tabular_bool
3604     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3605     \l_@@_exterior_arraycolsep_bool
3606     \l_@@_bar_at_end_of_pream_bool
3607   }
3608   { \skip_horizontal:n { - \col@sep } }
3609   \bool_if:NT \l_@@_code_before_bool
3610   {
3611     \hbox
3612     {
3613       \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.

```

3614 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3615 { \skip_horizontal:n { - \arraycolsep } }
3616 \pgfsys@markposition
3617 { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3618 \skip_horizontal:n { 0.5 \arrayrulewidth }
3619 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3620 { \skip_horizontal:N \arraycolsep }
3621 }
3622 }
3623 \pgfpicture
3624 \pgfrememberpicturepositiononpagetrue
3625 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3626 {

```

```

3627         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3628         {
3629             \pgfpoint
3630             { - 0.5 \arrayrulewidth - \arraycolsep }
3631             \c_zero_dim
3632         }
3633         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3634     }
3635     \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3636 \endpgfpicture
3637 }

3638 \bool_if:NT \g_@@_last_col_found_bool
3639 {
3640     \hbox_overlap_right:n
3641     {
3642         \skip_horizontal:N \g_@@_width_last_col_dim
3643         \skip_horizontal:N \colsep
3644         \bool_if:NT \l_@@_code_before_bool
3645         {
3646             \pgfsys@markposition
3647             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3648         }
3649         \pgfpicture
3650         \pgfrememberpicturepositiononpagetrue
3651         \pgfcoordinate
3652         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3653         \pgfpintorigin
3654         \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3655         \endpgfpicture
3656     }
3657 }
3658 }

3659 \cs_new_protected:Npn \@@_mark_position:n #1
3660 {
3661     \bool_if:NT \l_@@_code_before_bool
3662     {
3663         \hbox
3664         {
3665             \skip_horizontal:n { -0.5 \arrayrulewidth }
3666             \pgfsys@markposition { \@@_env: - col - #1 }
3667             \skip_horizontal:n { 0.5 \arrayrulewidth }
3668         }
3669     }
3670 }

3671 \cs_new_protected:Npn \@@_node_alias:n #1
3672 {
3673     \str_if_empty:NF \l_@@_name_str
3674     { \pgfnodelias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3675 }

```

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

```

3676 \tl_const:Nn \c_@@_preamble_first_col_tl
3677 {
3678     >
3679     {

```

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

```

3680      \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3681      \bool_gset_true:N \g_@@_after_col_zero_bool
3682      \@@_begin_of_row:
3683      \hbox_set:Nw \l_@@_cell_box
3684      \@@_math_toggle:
3685      \@@_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”.

```

3686      \int_compare:nNnT \c@iRow > \c_zero_int
3687      {
3688        \bool_lazy_or:nnT
3689        { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3690        { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3691        {
3692          \l_@@_code_for_first_col_tl
3693          \xglobal \colorlet { nicematrix-first-col } { . }
3694        }
3695      }
3696    }

```

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`.

```

3697      l
3698      <
3699      {
3700        \@@_math_toggle:
3701        \hbox_set_end:
3702        \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3703        \@@_adjust_size_box:
3704        \@@_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.

```

3705      \dim_gset:Nn \g_@@_width_first_col_dim
3706      { \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.

```

3707      \hbox_overlap_left:n
3708      {
3709        \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3710        \@@_node_cell:
3711        { \box_use_drop:N \l_@@_cell_box }
3712        \skip_horizontal:N \l_@@_left_delim_dim
3713        \skip_horizontal:N \l_@@_left_margin_dim
3714        \skip_horizontal:N \l_@@_extra_left_margin_dim
3715      }
3716      \bool_gset_false:N \g_@@_empty_cell_bool
3717      \skip_horizontal:n { -2 \col@sep }
3718    }
3719  }

```

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

```

3720  \tl_const:Nn \c_@@_preamble_last_col_tl
3721  {
3722    >
3723    {
3724      \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).

```

3725      \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

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

```

3726     \bool_gset_true:N \g_@@_last_col_found_bool
3727     \int_gincr:N \c@jCol
3728     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3729     \hbox_set:Nw \l_@@_cell_box
3730     \@@_math_toggle:
3731     \@@_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”.

```

3732     \int_compare:nNnT \c@iRow > \c_zero_int
3733     {
3734         \bool_lazy_or:nnT
3735         { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3736         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3737         {
3738             \l_@@_code_for_last_col_tl
3739             \xglobal \colorlet { nicematrix-last-col } { . }
3740         }
3741     }
3742 }
3743 1
3744 <
3745 {
3746     \@@_math_toggle:
3747     \hbox_set_end:
3748     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3749     \@@_adjust_size_box:
3750     \@@_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.

```

3751     \dim_gset:Nn \g_@@_width_last_col_dim
3752     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3753     \skip_horizontal:n { -2 \col@sep }

```

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

```

3754     \hbox_overlap_right:n
3755     {
3756         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3757         {
3758             \skip_horizontal:N \l_@@_right_delim_dim
3759             \skip_horizontal:N \l_@@_right_margin_dim
3760             \skip_horizontal:N \l_@@_extra_right_margin_dim
3761             \@@_node_cell:
3762         }
3763     }
3764     \bool_gset_false:N \g_@@_empty_cell_bool
3765 }
3766 }

```

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

```

3767 \NewDocumentEnvironment { NiceArray } { }
3768 {
3769     \bool_gset_false:N \g_@@_delims_bool
3770     \str_if_empty:NT \g_@@_name_env_str
3771     { \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).

```

3772     \NiceArrayWithDelims . .
3773 }
3774 { \endNiceArrayWithDelims }

```

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

```

3775 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3776 {
3777   \NewDocumentEnvironment { #1 NiceArray } { }
3778   {
3779     \bool_gset_true:N \g_@@_delims_bool
3780     \str_if_empty:NT \g_@@_name_env_str
3781     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3782     \@@_test_if_math_mode:
3783     \NiceArrayWithDelims #2 #3
3784   }
3785   { \endNiceArrayWithDelims }
3786 }
3787 \@@_def_env:NNN p ( )
3788 \@@_def_env:NNN b [ ]
3789 \@@_def_env:NNN B \{ \}
3790 \@@_def_env:NNN v \vert \vert
3791 \@@_def_env:NNN V \Vert \Vert

```

13 The environment `{NiceMatrix}` and its variants

13.1 Definition of `{pNiceMatrix}`

```

3792 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3793 {
3794   \bool_set_false:N \l_@@_preamble_bool
3795   \tl_clear:N \l_tmpa_tl
3796   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3797   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3798   \tl_put_right:Nn \l_tmpa_tl
3799   {
3800     *
3801     {
3802       \int_case:nnF \l_@@_last_col_int
3803       {
3804         { -2 } { \c@MaxMatrixCols }
3805         { -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).

```

3806       }
3807       { \int_eval:n { \l_@@_last_col_int - 1 } }
3808     }
3809     { #2 }
3810   }
3811   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3812   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3813 }
3814 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3815 \clist_map_inline:nn { p , b , B , v , V }
3816 {
3817   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3818   {
3819     \bool_gset_true:N \g_@@_delims_bool
3820     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3821     \int_if_zero:nT \l_@@_last_col_int
3822     {
3823       \bool_set_true:N \l_@@_last_col_without_value_bool
3824       \int_set:Nn \l_@@_last_col_int { -1 }

```



```

3825     }
3826     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3827     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3828   }
3829   { \use:c { end #1 NiceArray } }
3830 }

```

We define also an environment {NiceMatrix}

```

3831 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3832 {
3833   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3834   \int_if_zero:nT \l_@@_last_col_int
3835   {
3836     \bool_set_true:N \l_@@_last_col_without_value_bool
3837     \int_set:Nn \l_@@_last_col_int { -1 }
3838   }
3839   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3840   \bool_lazy_or:nnT
3841     \l_@@_except_borders_bool
3842     { \clist_if_empty_p:N \l_@@_vlines_clist }
3843     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3844   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3845 }
3846 { \endNiceArray }

```

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

```

3847 \cs_new_protected:Npn \@@_NotEmpty:
3848 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

13.2 The key renew-matrix

```

3849 \cs_set_protected:Npn \@@_renew_matrix:
3850 {
3851   \tl_map_inline:nn { pvVbB }
3852   { \RenewEnvironmentCopy { ##1matrix } { ##1NiceMatrix } }
3853 }

```

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

```

3854 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3855 {

```

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.

```

3856   \dim_compare:nNtT \l_@@_width_dim = \c_zero_dim
3857   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3858   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3859   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3860   \tl_if_empty:NF \l_@@_short_caption_tl
3861   {
3862     \tl_if_empty:NT \l_@@_caption_tl
3863     {
3864       \@@_error_or_warning:n { short-caption-without~caption }
3865       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3866     }
3867   }
3868   \tl_if_empty:NF \l_@@_label_tl
3869   {
3870     \tl_if_empty:NT \l_@@_caption_tl
3871     { \@@_error_or_warning:n { label~without~caption } }
3872   }

```

```

3873 \NewDocumentEnvironment { TabularNote } { b }
3874 {
3875     \bool_if:NTF \l_@@_in_code_after_bool
3876     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3877     {
3878         \tl_if_empty:NF \g_@@_tabularnote_tl
3879         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3880         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3881     }
3882 }
3883 { }
3884 \@@_settings_for_tabular:
3885 \NiceArray { #2 }
3886 }
3887 { \endNiceArray }
3888 \cs_new_protected:Npn \@@_settings_for_tabular:
3889 {
3890     \bool_set_true:N \l_@@_tabular_bool
3891     \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3892     \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3893     \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3894 }

3895 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3896 {
3897     \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3898     \dim_set:Nn \l_@@_width_dim { #1 }
3899     \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3900     \@@_settings_for_tabular:
3901     \NiceArray { #3 }
3902 }
3903 {
3904     \endNiceArray
3905     \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3906     { \@@_error:n { NiceTabularX~without~X } }
3907 }

3908 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3909 {
3910     \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3911     \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3912     \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3913     \@@_settings_for_tabular:
3914     \NiceArray { #3 }
3915 }
3916 { \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`).

```

3917 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3918 {
3919     \bool_lazy_all:nT
3920     {
3921         { \dim_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3922         { \l_@@_hvlines_bool }
3923         { ! \g_@@_delims_bool }

```

```

3924     { ! \l_@@_except_borders_bool }
3925   }
3926   {
3927     \bool_set_true:N \l_@@_except_borders_bool
3928     \clist_if_empty:NF \l_@@_corners_clist
3929     { \@@_error:n { hvlines,~rounded-corners-and~corners } }
3930     \tl_gput_right:Nn \g_@@_rules_tl
3931     {
3932       \@@_stroke_block:nnnnn
3933       {
3934         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3935         draw = \l_@@_rules_color_tl
3936       }
3937       { 1 } { 1 } { \int_use:N \c@iRow } { \int_use:N \c@jCol }
3938     }
3939   }
3940 }

```

```

3941 \cs_new_protected:Npn \@@_after_array:
3942 {

```

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`.

```

3943   \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3944   \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.

```

3945   \bool_if:NT \g_@@_last_col_found_bool
3946   { \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`.

```

3947   \bool_if:NT \l_@@_last_col_without_value_bool
3948   { \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.

```

3949   \bool_if:NT \l_@@_last_row_without_value_bool
3950   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3951   \tl_gput_right:Ne \g_@@_aux_tl
3952   {
3953     \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3954     {
3955       \int_use:N \l_@@_first_row_int ,
3956       \int_use:N \c@iRow ,
3957       \int_use:N \g_@@_row_total_int ,
3958       \int_use:N \l_@@_first_col_int ,
3959       \int_use:N \c@jCol ,
3960       \int_use:N \g_@@_col_total_int
3961     }
3962   }
3963   \clist_if_empty:NF \g_@@_cbic_clist \@@_create_blocks_in_col:

```

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`).

```

3964 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3965 {
3966   \tl_gput_right:Ne \g_@@_aux_tl
3967   {
3968     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3969     { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }
3970   }
3971 }
3972 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3973 {
3974   \tl_gput_right:Ne \g_@@_aux_tl
3975   {
3976     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3977     { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
3978     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3979     { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
3980   }
3981 }

```

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

```

3982 \@@_create_diag_nodes:

```

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

```

3983 \pgfpicture
3984 \@@_create_aliases_last:
3985 \str_if_empty:NF \l_@@_name_str \@@_create_alias_nodes:
3986 \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.

```

3987 \bool_if:NT \l_@@_parallelize_diags_bool
3988 {
3989   \int_gzero:N \g_@@_ddots_int
3990   \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.

```

3991   \dim_gzero:N \g_@@_delta_x_one_dim
3992   \dim_gzero:N \g_@@_delta_y_one_dim
3993   \dim_gzero:N \g_@@_delta_x_two_dim
3994   \dim_gzero:N \g_@@_delta_y_two_dim
3995 }
3996 \bool_set_false:N \l_@@_initial_open_bool
3997 \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.

```

3998 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

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

```

3999 \@@_draw_dotted_lines:

```

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

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.

```

4000   \clist_if_empty:NF \l_@@_corners_clist
4001   {
4002     \bool_if:NTF \l_@@_no_cell_nodes_bool
4003     { \@@_error:n { corners~with~no~cell~nodes } }
4004     \@@_compute_corners:
4005   }

```

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

```

4006   \seq_gconcat:NNN \g_@@_pos_of_blocks_seq
4007   \g_@@_pos_of_blocks_seq
4008   \g_@@_future_pos_of_blocks_seq
4009   \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-*}`).

```

4010   \@@_adjust_pos_of_blocks_seq:
4011   \@@_deal_with_rounded_corners:
4012   \legacy_if:nF { measuring@ } \@@_draw_rules:
4013   \tl_gclear:N \g_@@_rules_tl

```

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

```

4014   \IfPackageLoadedT { tikz }
4015   {
4016     \tikzset
4017     {
4018       every~picture / .style =
4019       {
4020         overlay ,
4021         remember~picture ,
4022         name~prefix = \@@_env: -
4023       }
4024     }
4025   }
4026   \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
4027   \cs_set_eq:NN \SubMatrix \@@_SubMatrix
4028   \cs_set_eq:NN \UnderBrace \@@_UnderBrace
4029   \cs_set_eq:NN \OverBrace \@@_OverBrace
4030   \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
4031   \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
4032   \cs_set_eq:NN \line \@@_line
4033

```

The LaTeX-style boolean `\ifmeasuring@` is used by `amsmath` during the phase of measure in environments such as `{align}`, etc.

```

4034   \legacy_if:nF { measuring@ } \g_@@_pre_code_after_tl
4035   \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.

```

4036   \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).

```

4037   \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`).

```
4038 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
4039 { \@@_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:`.

```
4040 \bool_set_true:N \l_@@_in_code_after_bool
4041 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4042 \scan_stop:
4043 \tl_gclear:N \g_nicematrix_code_after_tl
4044 \clist_if_empty:NF \g_@@_col_with_trees_clist \@@_draw_trees:
4045 \group_end:
```

`\g_@@_pre_code_before_tl` is for instructions in the cells of the array such as `\rowcolor` and `\cellcolor`. These instructions will be written on the aux file to be added to the code-before in the next run.

```
4046 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
4047 \tl_if_empty:NF \g_@@_pre_code_before_tl
4048 {
4049   \tl_gput_right:Ne \g_@@_aux_tl
4050   {
4051     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
4052     { \exp_not:o \g_@@_pre_code_before_tl }
4053   }
4054   \tl_gclear:N \g_@@_pre_code_before_tl
4055 }
4056 \tl_if_empty:NF \g_nicematrix_code_before_tl
4057 {
4058   \tl_gput_right:Ne \g_@@_aux_tl
4059   {
4060     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
4061     { \exp_not:o \g_nicematrix_code_before_tl }
4062   }
4063   \tl_gclear:N \g_nicematrix_code_before_tl
4064 }

4065 \str_gclear:N \g_@@_name_env_str
4066 \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the array¹³. This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able to use color for the rules of the array when `colortbl` is *not* loaded. That's why we do the following instruction which is in the patch of the end of arrays done by `colortbl`.

```
4067 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
4068 }

4069 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
4070 {
4071   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
4072   \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }
```

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

```
4073 \dim_set:Nn \l_@@_xdots_shorten_start_dim
4074 { 0.6 \l_@@_xdots_shorten_start_dim }
```

¹³e.g. `\color[rgb]{0.5,0.5,0}`

```

4075 \dim_set:Nn \l_@@_xdots_shorten_end_dim
4076 { 0.6 \l_@@_xdots_shorten_end_dim }
4077 }

```

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

```

4078 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4079 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

4080 \cs_new_protected:Npn \@@_create_alias_nodes:
4081 {
4082   \int_step_inline:nn \c@iRow
4083   {
4084     \pgfnodealias
4085     { \l_@@_name_str - ##1 - last }
4086     { \@@_env: - ##1 - \int_use:N \c@jCol }
4087   }
4088   \int_step_inline:nn \c@jCol
4089   {
4090     \pgfnodealias
4091     { \l_@@_name_str - last - ##1 }
4092     { \@@_env: - \int_use:N \c@iRow - ##1 }
4093   }
4094   \pgfnodealias
4095   { \l_@@_name_str - last - last }
4096   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
4097 }

```

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

```

4098 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4099 {
4100   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4101   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4102 }

```

The following command must *not* be protected.

```

4103 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4104 {
4105   { #1 }
4106   { #2 }
4107   {
4108     \int_compare:nNnTF { #3 } > { 98 }
4109     { \int_use:N \c@iRow }
4110     { #3 }
4111   }
4112   {
4113     \int_compare:nNnTF { #4 } > { 98 }
4114     { \int_use:N \c@jCol }
4115     { #4 }
4116   }
4117   { #5 }
4118 }

```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible”. That’s why we have to define the adequate version of `\@@_draw_dotted_lines`: whether TikZ is loaded or not (in that case, only PGF is loaded).

```

4119 \AtBeginDocument
4120 {
4121   \cs_new_protected:Npn \@@_draw_dotted_lines:
4122   {
4123     \c_@@_pgfortikzpicture_tl
4124     \@@_draw_dotted_lines_i:
4125     \c_@@_endpgfortikzpicture_tl
4126   }
4127 }
```

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

```

4128 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4129 {
4130   \pgfrememberpicturepositiononpagetrue
4131   \pgf@relevantforpicturesizefalse
4132   \g_@@_HVdotsfor_lines_tl
4133   \g_@@_Vdots_lines_tl
4134   \g_@@_Ddots_lines_tl
4135   \g_@@_Iddots_lines_tl
4136   \g_@@_Cdots_lines_tl
4137   \g_@@_Ldots_lines_tl
4138 }

4139 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4140 {
4141   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4142   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4143 }
```

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

```

4144 \pgfdeclareshape { @@_diag_node }
4145 {
4146   \savedanchor { \five }
4147   {
4148     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4149     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4150   }
4151   \anchor { 5 } { \five }
4152   \anchor { center } { \pgfpointorigin }
4153   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4154   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4155   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4156   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4157   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4158   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4159   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4160   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4161   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4162   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4163 }
```

The following command creates the diagonal nodes (in fact, if the matrix is not a square matrix, not all the nodes are on the diagonal).

```

4164 \cs_new_protected:Npn \@@_create_diag_nodes:
4165 {
4166   \pgfpicture
```



```

4167 \pgfrememberpicturepositiononpagetrue
4168 \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4169 {
4170   \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4171   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4172   \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4173   \dim_set_eq:NN \l_tmpb_dim \pgf@y
4174   \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4175   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4176   \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4177   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4178   \pgftransformshift { \pgfpoint \l_tmpa_dim \l_tmpb_dim }

```

Now, `\l_tmpa_dim` and `\l_tmpb_dim` become the width and the height of the node (of shape `@@_diag_node`) that we will construct.

```

4179   \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4180   \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4181   \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4182   \str_if_empty:NF \l_@@_name_str
4183   { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4184 }

```

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

```

4185   \int_set:Nn \l_tmpa_int { 1 + \int_max:nn \c@iRow \c@jCol } % modified
4186   \@@_qpoint:n { row - \int_min:nn \l_tmpa_int { \c@iRow + 1 } }
4187   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4188   \@@_qpoint:n { col - \int_min:nn \l_tmpa_int { \c@jCol + 1 } }
4189   \pgfcoordinate
4190   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4191   \pgfnodealias
4192   { \@@_env: - last }
4193   { \@@_env: - \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
4194   \str_if_empty:NF \l_@@_name_str
4195   {
4196     \pgfnodealias
4197     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4198     { \@@_env: - \int_use:N \l_tmpa_int }
4199     \pgfnodealias
4200     { \l_@@_name_str - last }
4201     { \@@_env: - last }
4202   }
4203   \endpgfpicture
4204 }

```

16 We draw the dotted lines

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

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

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

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;

- the third argument is the x -value of the orientation vector of the line;
- the fourth argument is the y -value of the orientation vector of the line.

This command computes:

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

We provide first a version in the L3 syntax, and, then a version slightly more efficient.

```
\cs_new_protected:Npn \l_@@_find_extremities:nnnn #1 #2 #3 #4
{
  \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
  \int_set:Nn \l_@@_initial_i_int { #1 }
  \int_set:Nn \l_@@_initial_j_int { #2 }
  \int_set:Nn \l_@@_final_i_int { #1 }
  \int_set:Nn \l_@@_final_j_int { #2 }
  \bool_set_false:N \l_@@_stop_loop_bool
  \bool_do_until:Nn \l_@@_stop_loop_bool
  {
    \int_add:Nn \l_@@_final_i_int { #3 }
    \int_add:Nn \l_@@_final_j_int { #4 }
    \bool_set_false:N \l_@@_final_open_bool
    \int_compare:nNnTF { \l_@@_final_i_int } > { \l_@@_row_max_int }
    {
      \int_compare:nNnTF { #3 } = { 1 }
      { \bool_set_true:N \l_@@_final_open_bool }
      {
        \int_compare:nNnT { \l_@@_final_j_int } > { \l_@@_col_max_int }
        { \bool_set_true:N \l_@@_final_open_bool }
      }
    }
  }
  {
    \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
    {
      \int_compare:nNnT { #4 } = { -1 }
      { \bool_set_true:N \l_@@_final_open_bool }
    }
    {
      \int_compare:nNnT { \l_@@_final_j_int } > { \l_@@_col_max_int }
      {
        \int_compare:nNnT { #4 } = { 1 }
        { \bool_set_true:N \l_@@_final_open_bool }
      }
    }
  }
}
\bool_if:NTF \l_@@_final_open_bool
{
  \int_sub:Nn \l_@@_final_i_int { #3 }
  \int_sub:Nn \l_@@_final_j_int { #4 }
  \bool_set_true:N \l_@@_stop_loop_bool
}
{
  \cs_if_exist:cTF
  {
    @@ _ dotted _
    \int_use:N \l_@@_final_i_int -
```

```

\int_use:N \l_@@_final_j_int
}
{
\int_sub:Nn \l_@@_final_i_int { #3 }
\int_sub:Nn \l_@@_final_j_int { #4 }
\bool_set_true:N \l_@@_final_open_bool
\bool_set_true:N \l_@@_stop_loop_bool
}
{
\cs_if_exist:cTF
{
pgf @ sh @ ns @ \@@_env:
- \int_use:N \l_@@_final_i_int
- \int_use:N \l_@@_final_j_int
}
{ \bool_set_true:N \l_@@_stop_loop_bool }
{
\cs_set_nopar:cpn
{
@@ _ dotted _
\int_use:N \l_@@_final_i_int -
\int_use:N \l_@@_final_j_int
}
{ }
}
}
}
}
\bool_set_false:N \l_@@_stop_loop_bool
\int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
\bool_do_until:Nn \l_@@_stop_loop_bool
{
\int_sub:Nn \l_@@_initial_i_int { #3 }
\int_sub:Nn \l_@@_initial_j_int { #4 }
\bool_set_false:N \l_@@_initial_open_bool
\int_compare:nNnTF { \l_@@_initial_i_int } < { \l_@@_row_min_int }
{
\int_compare:nNnTF { #3 } = { 1 }
{ \bool_set_true:N \l_@@_initial_open_bool }
{
\int_compare:nNnT { \l_@@_initial_j_int } = { \l_tmpa_int }
{ \bool_set_true:N \l_@@_initial_open_bool }
}
}
}
{
\int_compare:nNnTF { \l_@@_initial_j_int } < { \l_@@_col_min_int }
{
\int_compare:nNnT { #4 } = { 1 }
{ \bool_set_true:N \l_@@_initial_open_bool }
}
{
\int_compare:nNnT { \l_@@_initial_j_int } > { \l_@@_col_max_int }
{
\int_compare:nNnT { #4 } = { -1 }
{ \bool_set_true:N \l_@@_initial_open_bool }
}
}
}
}
\bool_if:NNTF \l_@@_initial_open_bool
{
\int_add:Nn \l_@@_initial_i_int { #3 }
\int_add:Nn \l_@@_initial_j_int { #4 }
\bool_set_true:N \l_@@_stop_loop_bool
}

```

```

    }
    {
      \cs_if_exist:cTF
      {
        @@ _ dotted _
        \int_use:N \l_@@_initial_i_int -
        \int_use:N \l_@@_initial_j_int
      }
      {
        \int_add:Nn \l_@@_initial_i_int { #3 }
        \int_add:Nn \l_@@_initial_j_int { #4 }
        \bool_set_true:N \l_@@_initial_open_bool
        \bool_set_true:N \l_@@_stop_loop_bool
      }
    }
    {
      \cs_if_exist:cTF
      {
        {
          pgf @ sh @ ns @ \@@_env:
          - \int_use:N \l_@@_initial_i_int
          - \int_use:N \l_@@_initial_j_int
        }
        { \bool_set_true:N \l_@@_stop_loop_bool }
        {
          \cs_set_nopar:cpn
          {
            @@ _ dotted _
            \int_use:N \l_@@_initial_i_int -
            \int_use:N \l_@@_initial_j_int
          }
          { }
        }
      }
    }
  }
}
\seq_gput_right:Ne \g_@@_pos_of_xdots_seq
{
  { \int_use:N \l_@@_initial_i_int }
  { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
  { \int_use:N \l_@@_final_i_int }
  { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
  { }
}
}

```

The following version is slightly more efficient.

```

4205 \cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
4206 {

```

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

```

4207 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4208 \l_@@_initial_i_int = #1
4209 \l_@@_initial_j_int = #2
4210 \l_@@_final_i_int = #1
4211 \l_@@_final_j_int = #2

```

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

```

4212 \let \l_@@_stop_loop_bool \c_false_bool
4213 \bool_do_until:Nn \l_@@_stop_loop_bool
4214 {

```

We test if we are still in the matrix.

```

4215     \advance \l_@@_final_i_int by #3
4216     \advance \l_@@_final_j_int by #4
4217     \let \l_@@_final_open_bool \c_false_bool
4218     \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4219       \if_int_compare:w #3 = \c_one_int
4220         \let \l_@@_final_open_bool \c_true_bool
4221       \else:
4222         \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4223           \let \l_@@_final_open_bool \c_true_bool
4224         \fi:
4225       \fi:
4226     \else:
4227       \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4228         \if_int_compare:w #4 = -1
4229           \let \l_@@_final_open_bool \c_true_bool
4230         \fi:
4231       \else:
4232         \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4233           \if_int_compare:w #4 = \c_one_int
4234             \let \l_@@_final_open_bool \c_true_bool
4235           \fi:
4236         \fi:
4237       \fi:
4238     \fi:
4239     \bool_if:NTF \l_@@_final_open_bool

```

If we are outside the matrix, we have found the extremity of the dotted line and it's an *open* extremity.

```

4240     {

```

We do a step backwards.

```

4241         \advance \l_@@_final_i_int by - #3
4242         \advance \l_@@_final_j_int by - #4
4243         \let \l_@@_stop_loop_bool \c_true_bool
4244     }

```

If we are in the matrix, we test whether the cell is empty. If it's not the case, we stop the loop because we have found the correct values for `\l_@@_final_i_int` and `\l_@@_final_j_int`.

```

4245     {
4246       \cs_if_exist:cTF
4247       {
4248         @@ _ dotted _
4249         \int_use:N \l_@@_final_i_int -
4250         \int_use:N \l_@@_final_j_int
4251       }
4252       {
4253         \advance \l_@@_final_i_int by - #3
4254         \advance \l_@@_final_j_int by - #4
4255         \let \l_@@_final_open_bool \c_true_bool
4256         \let \l_@@_stop_loop_bool \c_true_bool
4257       }
4258       {
4259         \cs_if_exist:cTF
4260         {
4261           pgf @ sh @ ns @ \@@_env:
4262           - \int_use:N \l_@@_final_i_int
4263           - \int_use:N \l_@@_final_j_int
4264         }
4265         { \let \l_@@_stop_loop_bool \c_true_bool }

```

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

```

4266         {
4267             \cs_set_nopar:cpn
4268             {
4269                 @@ _ dotted _
4270                 \int_use:N \l_@@_final_i_int -
4271                 \int_use:N \l_@@_final_j_int
4272             }
4273             { }
4274         }
4275     }
4276 }
4277 }

```

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

```

4278 \let \l_@@_stop_loop_bool \c_false_bool

```

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

```

4279 \l_tmpa_int = \l_@@_col_min_int
4280 \advance \l_tmpa_int by -1
4281 \bool_do_until:Nn \l_@@_stop_loop_bool
4282 {
4283     \advance \l_@@_initial_i_int by - #3
4284     \advance \l_@@_initial_j_int by - #4
4285     \let \l_@@_initial_open_bool \c_false_bool

```

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

```

4286 \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4287 \if_int_compare:w #3 = \c_one_int
4288 \let \l_@@_initial_open_bool \c_true_bool
4289 \else:

```

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

```

4290 \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4291 \let \l_@@_initial_open_bool \c_true_bool
4292 \fi:
4293 \fi:
4294 \else:
4295 \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4296 \if_int_compare:w #4 = \c_one_int
4297 \let \l_@@_initial_open_bool \c_true_bool
4298 \fi:
4299 \else:
4300 \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4301 \if_int_compare:w #4 = -1
4302 \let \l_@@_initial_open_bool \c_true_bool
4303 \fi:
4304 \fi:
4305 \fi:
4306 \fi:
4307 \bool_if:NTF \l_@@_initial_open_bool
4308 {
4309     \advance \l_@@_initial_i_int by #3
4310     \advance \l_@@_initial_j_int by #4
4311     \let \l_@@_stop_loop_bool \c_true_bool
4312 }
4313 {
4314     \cs_if_exist:cTF
4315     {
4316         @@ _ dotted _
4317         \int_use:N \l_@@_initial_i_int -

```

```

4318         \int_use:N \l_@@_initial_j_int
4319     }
4320     {
4321         \advance \l_@@_initial_i_int by #3
4322         \advance \l_@@_initial_j_int by #4
4323         \let \l_@@_initial_open_bool \c_true_bool
4324         \let \l_@@_stop_loop_bool \c_true_bool
4325     }
4326     {
4327         \cs_if_exist:cTF
4328         {
4329             pgf @ sh @ ns @ \@@_env:
4330             - \int_use:N \l_@@_initial_i_int
4331             - \int_use:N \l_@@_initial_j_int
4332         }
4333         { \let \l_@@_stop_loop_bool \c_true_bool }
4334         {
4335             \cs_set_nopar:cpn
4336             {
4337                 @@ _ dotted _
4338                 \int_use:N \l_@@_initial_i_int -
4339                 \int_use:N \l_@@_initial_j_int
4340             }
4341             { }
4342         }
4343     }
4344 }
4345 }

```

We remind the rectangle described by all the dotted lines in order to respect the corresponding virtual “block” when drawing the horizontal and vertical rules.

```

4346     \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4347     {
4348         { \int_use:N \l_@@_initial_i_int }

```

Be careful: with `\Iddots`, `\l_@@_final_j_int` is inferior to `\l_@@_initial_j_int`. That’s why we use `\int_min:nn` and `\int_max:nn`.

```

4349         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4350         { \int_use:N \l_@@_final_i_int }
4351         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4352         { }
4353     }
4354 }

```

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

```

4355 \cs_new_protected:Npn \@@_open_shorten:
4356 {
4357     \bool_if:NT \l_@@_initial_open_bool
4358     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4359     \bool_if:NT \l_@@_final_open_bool
4360     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4361 }

```

The following command (*when it will be written*) will set the four counters `\l_@@_row_min_int`, `\l_@@_row_max_int`, `\l_@@_col_min_int` and `\l_@@_col_max_int` to the intersections of the sub-matrices which contains the cell of row #1 and column #2. As of now, it’s only the whole array (excepted exterior rows and columns).

```

4362 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2

```

```

4363 {
4364   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4365   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4366   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4367   \int_set_eq:NN \l_@@_col_max_int \c@jCol

```

We do a loop over all the submatrices specified in the `code-before`. We have stored the position of all those submatrices in `\g_@@_submatrix_seq`.

```

4368   \seq_if_empty:NF \g_@@_submatrix_seq
4369   {
4370     \seq_map_inline:Nn \g_@@_submatrix_seq
4371     { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4372   }
4373 }

```

`#1` and `#2` are the numbers of row and columns of the cell where the command of dotted line (ex.: `\Vdots`) has been issued. `#3`, `#4`, `#5` and `#6` are the specification (in i and j) of the submatrix we are analyzing.

Here is the programming of that command with the the standard syntax of L3.

```

\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
{
  \bool_if:nT
  {
    \int_compare_p:n { #3 <= #1 <= #5 }
    &&
    \int_compare_p:n { #4 <= #2 <= #6 }
  }
  {
    \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
    \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
    \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
    \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
  }
}

```

However, for efficiency, we will use the following version.

```

4374 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4375 {
4376   \if_int_compare:w #3 > #1
4377   \else:
4378     \if_int_compare:w #1 > #5
4379     \else:
4380       \if_int_compare:w #4 > #2
4381       \else:
4382         \if_int_compare:w #2 > #6
4383         \else:
4384           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4385           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4386           \if_int_compare:w \l_@@_row_max_int > #5 \l_@@_row_max_int = #5 \fi:
4387           \if_int_compare:w \l_@@_col_max_int > #6 \l_@@_col_max_int = #6 \fi:
4388         \fi:
4389       \fi:
4390     \fi:
4391   \fi:
4392 }

4393 \cs_new_protected:Npn \@@_set_initial_coords:
4394 {
4395   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4396   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4397 }
4398 \cs_new_protected:Npn \@@_set_final_coords:

```



```

4399 {
4400   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4401   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4402 }
4403 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4404 {
4405   \pgfpointanchor
4406   {
4407     \@@_env:
4408     - \int_use:N \l_@@_initial_i_int
4409     - \int_use:N \l_@@_initial_j_int
4410   }
4411   { #1 }
4412   \@@_set_initial_coords:
4413 }
4414 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4415 {
4416   \pgfpointanchor
4417   {
4418     \@@_env:
4419     - \int_use:N \l_@@_final_i_int
4420     - \int_use:N \l_@@_final_j_int
4421   }
4422   { #1 }
4423   \@@_set_final_coords:
4424 }
4425 \cs_new_protected:Npn \@@_open_x_initial_dim:
4426 {
4427   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4428   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4429   {
4430     \cs_if_exist:cT
4431     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4432     {
4433       \pgfpointanchor
4434       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4435       { west }
4436       \dim_set:Nn \l_@@_x_initial_dim
4437       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4438     }
4439   }

```

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

```

4440   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4441   {
4442     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4443     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4444     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4445   }
4446 }
4447 \cs_new_protected:Npn \@@_open_x_final_dim:
4448 {
4449   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4450   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4451   {
4452     \cs_if_exist:cT
4453     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4454     {
4455       \pgfpointanchor
4456       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4457       { east }
4458       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4459       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }

```

```

4460     }
4461 }

```

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

```

4462 \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4463 {
4464   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4465   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4466   \dim_sub:Nn \l_@@_x_final_dim \col@sep
4467 }
4468 }

```

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

4469 \cs_new_protected:Npn \@@_draw_Ldots:nmn #1 #2 #3
4470 {
4471   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4472   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4473   {
4474     \@@_find_extremities:nnnn { #1 } { #2 } 0 1

```

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

```

4475     \bool_if:NT \g_@@_aux_found_bool
4476     {
4477       {
4478         \@@_open_shorten:
4479         \int_if_zero:nTF { #1 }
4480         { \color { nicematrix-first-row } }
4481         {

```

We remind that, when there is a “last row” `\l_@@_last_row_int` will always be (after the construction of the array) the number of that “last row” even if the option `last-row` has been used without value.

```

4482         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4483         { \color { nicematrix-last-row } }
4484       }
4485       \keys_set:nn { nicematrix / xdots } { #3 }
4486       \@@_color:o \l_@@_xdots_color_tl
4487       \@@_actually_draw_Ldots:
4488     }
4489   }
4490 }
4491 }

```

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`.

```

4492 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4493 {
4494   \bool_if:NTF \l_@@_initial_open_bool
4495   {
4496     \@@_open_x_initial_dim:

```

```

4497     \l_@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4498     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4499   }
4500   { \l_@@_set_initial_coords_from_anchor:n { base~east } }
4501 \bool_if:NTF \l_@@_final_open_bool
4502 {
4503   \l_@@_open_x_final_dim:
4504   \l_@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4505   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4506 }
4507 { \l_@@_set_final_coords_from_anchor:n { base~west } }

```

Now the case of a `\Hdotsfor` (or when there is only a `\Ldots`) in the “last row” (that case will probably arise when the final user draws an arrow to indicate the number of columns of the matrix). In the “first row”, we don’t need any adjustment.

```

4508   \bool_lazy_all:nTF
4509   {
4510     \l_@@_initial_open_bool
4511     \l_@@_final_open_bool
4512     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4513   }
4514   {
4515     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4516     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4517   }

```

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

```

4518   {
4519     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4520     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4521   }
4522   \l_@@_draw_line:
4523 }

```

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

4524 \cs_new_protected:Npn \l_@@_draw_Cdots:nnn #1 #2 #3
4525 {
4526   \l_@@_adjust_to_submatrix:nn { #1 } { #2 }
4527   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4528   {
4529     \l_@@_find_extremities:nnnn { #1 } { #2 } 0 1

```

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

```

4530   \bool_if:NNT \g_@@_aux_found_bool
4531   {
4532     {
4533       \l_@@_open_shorten:
4534       \int_if_zero:nTF { #1 }
4535       { \color { nicematrix-first-row } }
4536     }

```

We remind that, when there is a “last row” `\l_@@_last_row_int` will always be (after the construction of the array) the number of that “last row” even if the option `last-row` has been used without value.

```

4537       \int_compare:nNnT { #1 } = \l_@@_last_row_int
4538       { \color { nicematrix-last-row } }
4539     }
4540     \keys_set:nn { nicematrix / xdots } { #3 }
4541     \l_@@_color:o \l_@@_xdots_color_tl
4542     \l_@@_actually_draw_Cdots:
4543   }

```

```

4544     }
4545   }
4546 }

```

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

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

```

4547 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4548 {
4549   \bool_if:NTF \l_@@_initial_open_bool
4550     \@@_open_x_initial_dim:
4551     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4552   \bool_if:NTF \l_@@_final_open_bool
4553     \@@_open_x_final_dim:
4554     { \@@_set_final_coords_from_anchor:n { mid-west } }
4555   \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4556   {
4557     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4558     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4559     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4560     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4561     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4562   }
4563   {
4564     \bool_if:NT \l_@@_initial_open_bool
4565     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4566     \bool_if:NT \l_@@_final_open_bool
4567     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4568   }
4569   \@@_draw_line:
4570 }
4571 \cs_new_protected:Npn \@@_open_y_initial_dim:
4572 {
4573   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4574   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4575   {
4576     \cs_if_exist:cT
4577     { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4578     {
4579       \pgfpointanchor
4580       { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4581       { north }
4582       \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4583       { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4584     }
4585   }
4586   \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4587   {
4588     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4589     \dim_set:Nn \l_@@_y_initial_dim
4590     {
4591       \fp_to_dim:n
4592       {
4593         \pgf@y

```

```

4594         + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4595     }
4596 }
4597 }
4598 }
4599 \cs_new_protected:Npn \@@_open_y_final_dim:
4600 {
4601     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4602     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4603     {
4604         \cs_if_exist:cT
4605         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4606         {
4607             \pgfpointanchor
4608             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4609             { south }
4610             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4611             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4612         }
4613     }
4614     \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4615     {
4616         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4617         \dim_set:Nn \l_@@_y_final_dim
4618         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4619     }
4620 }

```

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.

```

4621 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4622 {
4623     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4624     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4625     {
4626         \@@_find_extremities:nnnn { #1 } { #2 } 1 0

```

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

```

4627     \bool_if:NT \g_@@_aux_found_bool
4628     {
4629         {
4630             \@@_open_shorten:
4631             \int_if_zero:nTF { #2 }
4632             { \color { nicematrix-first-col } }
4633             {
4634                 \int_compare:nNnT { #2 } = \l_@@_last_col_int
4635                 { \color { nicematrix-last-col } }
4636             }
4637             \keys_set:nn { nicematrix / xdots } { #3 }
4638             \@@_color:o \l_@@_xdots_color_tl
4639             \bool_if:NTF \l_@@_Vbrace_bool
4640             \@@_actually_draw_Vbrace:
4641             \@@_actually_draw_Vdots:
4642         }
4643     }
4644 }
4645 }

```

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.

```

4646 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4647 {
4648   \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4649   \@@_actually_draw_Vdots_i:
4650   \@@_actually_draw_Vdots_ii:
4651   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4652   \@@_draw_line:
4653 }

```

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

```

4654 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4655 {
4656   \@@_open_y_initial_dim:
4657   \@@_open_y_final_dim:
4658   \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4659 {
4660   \@@_qpoint:n { col - 1 }
4661   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4662   \dim_sub:Nn \l_@@_x_initial_dim
4663   { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4664 }
4665 {
4666   \bool_lazy_and:nnTF
4667   { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4668   { \int_compare_p:nNn \l_@@_initial_j_int = \g_@@_col_total_int }

```

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

```

4669 {
4670   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4671   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4672   \dim_add:Nn \l_@@_x_initial_dim
4673   { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4674 }

```

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

```

4675 {
4676   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4677   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4678   \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4679   \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4680 }
4681 }
4682 }

```

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.

```

4683 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4684 {
4685   \bool_set_false:N \l_tmpa_bool
4686   \bool_if:NF \l_@@_initial_open_bool

```

```

4687 {
4688   \bool_if:NF \l_@@_final_open_bool
4689   {
4690     \@@_set_initial_coords_from_anchor:n { south~west }
4691     \@@_set_final_coords_from_anchor:n { north~west }
4692     \bool_set:Nn \l_tmpa_bool
4693     { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4694   }
4695 }

```

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

```

4696 \bool_if:NTF \l_@@_initial_open_bool
4697 {
4698   \@@_open_y_initial_dim:
4699   \@@_set_final_coords_from_anchor:n { north }
4700   \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4701 }
4702 {
4703   \@@_set_initial_coords_from_anchor:n { south }
4704   \bool_if:NTF \l_@@_final_open_bool
4705   \@@_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.

```

4706 {
4707   \@@_set_final_coords_from_anchor:n { north }
4708   \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4709   {
4710     \dim_set:Nn \l_@@_x_initial_dim
4711     {
4712       \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4713       \l_@@_x_initial_dim \l_@@_x_final_dim
4714     }
4715   }
4716 }
4717 }
4718 }

```

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`.

```

4719 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4720 {
4721   \bool_if:NTF \l_@@_initial_open_bool
4722   \@@_open_y_initial_dim:
4723   { \@@_set_initial_coords_from_anchor:n { south } }
4724   \bool_if:NTF \l_@@_final_open_bool
4725   \@@_open_y_final_dim:
4726   { \@@_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.

```

4727 \int_if_zero:nTF \l_@@_initial_j_int
4728 {
4729     \@@_qpoint:n { col - 1 }
4730     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4731     \dim_sub:Nn \l_@@_x_initial_dim
4732     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4733 }

```

Elsewhere, the brace must be drawn left flush.

```

4734 {
4735     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4736     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4737     \dim_add:Nn \l_@@_x_initial_dim
4738     { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4739 }

```

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

```

4740 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4741 \@@_draw_line:
4742 }

```

```

4743 \cs_new:Npn \@@_colsep:
4744 { \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.

```

4745 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4746 {
4747     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4748     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4749     {
4750         \@@_find_extremities:nnnn { #1 } { #2 } { 1 } { 1 }

```

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

```

4751     \bool_if:NT \g_@@_aux_found_bool
4752     {
4753         {
4754             \@@_open_shorten:
4755             \keys_set:nn { nicematrix / xdots } { #3 }
4756             \@@_color:o \l_@@_xdots_color_tl
4757             \@@_actually_draw_Ddots:
4758         }
4759     }
4760 }
4761 }

```

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`.


```

4762 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4763 {
4764   \bool_if:NTF \l_@@_initial_open_bool
4765   {
4766     \@@_open_y_initial_dim:
4767     \@@_open_x_initial_dim:
4768   }
4769   { \@@_set_initial_coords_from_anchor:n { south-east } }
4770   \bool_if:NTF \l_@@_final_open_bool
4771   {
4772     \@@_open_x_final_dim:
4773     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4774   }
4775   { \@@_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.

```

4776   \bool_if:NT \l_@@_parallelize_diags_bool
4777   {
4778     \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).

```

4779     \int_compare:nNnTF \g_@@_ddots_int = 1

```

If the diagonal line is the first one, we have no adjustment of the line to do but we store the Δ_x and the Δ_y of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```

4780     {
4781       \dim_gset:Nn \g_@@_delta_x_one_dim
4782       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4783       \dim_gset:Nn \g_@@_delta_y_one_dim
4784       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4785     }

```

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`.

```

4786     {
4787       \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4788       {
4789         \dim_set:Nn \l_@@_y_final_dim
4790         {
4791           \l_@@_y_initial_dim +
4792           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4793           \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4794         }
4795       }
4796     }
4797   }
4798   \@@_draw_line:
4799 }

```

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.

```

4800 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4801 {
4802   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4803   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4804   {
4805     \@@_find_extremities:nnnn { #1 } { #2 } { 1 } { -1 }

```

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

```

4806     \bool_if:NT \g_@@_aux_found_bool

```

```

4807     {
4808     {
4809         \@@_open_shorten:
4810         \keys_set:nn { nicematrix / xdots } { #3 }
4811         \@@_color:o \l_@@_xdots_color_tl
4812         \@@_actually_draw_Iddots:
4813     }
4814 }
4815 }
4816 }

```

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.`

```

4817 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4818 {
4819     \bool_if:NTF \l_@@_initial_open_bool
4820     {
4821         \@@_open_y_initial_dim:
4822         \@@_open_x_initial_dim:
4823     }
4824     { \@@_set_initial_coords_from_anchor:n { south-west } }
4825     \bool_if:NTF \l_@@_final_open_bool
4826     {
4827         \@@_open_y_final_dim:
4828         \@@_open_x_final_dim:
4829     }
4830     { \@@_set_final_coords_from_anchor:n { north-east } }
4831     \bool_if:NT \l_@@_parallelize_diags_bool
4832     {
4833         \int_gincr:N \g_@@_iddots_int
4834         \int_compare:nNnTF \g_@@_iddots_int = 1
4835         {
4836             \dim_gset:Nn \g_@@_delta_x_two_dim
4837             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4838             \dim_gset:Nn \g_@@_delta_y_two_dim
4839             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4840         }
4841         {
4842             \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4843             {
4844                 \dim_set:Nn \l_@@_y_final_dim
4845                 {
4846                     \l_@@_y_initial_dim +
4847                     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4848                     \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4849                 }
4850             }
4851         }
4852     }
4853     \@@_draw_line:
4854 }

```

17 The actual instructions for drawing the dotted lines with TikZ

The command `\@@_draw_line:` 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`

```

4855 \cs_new_protected:Npn \@@_draw_line:
4856 {
4857   \pgfrememberpicturepositiononpagetrue
4858   \pgf@relevantforpicturesizefalse
4859   \bool_lazy_or:nnTF
4860     \l_@@_dotted_bool
4861     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4862     \@@_draw_standard_dotted_line:
4863     \@@_draw_unstandard_dotted_line:
4864 }

```

We have to do a special construction with `\exp_args:No` to be able to put in the list of options in the correct place in the TikZ instruction.

```

4865 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4866 {
4867   \begin { scope }
4868     \@@_draw_unstandard_dotted_line:o
4869     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4870 }

```

We have used the fact that, in PGF, a color name can be put directly in a list of options (that's why we have put directly `\l_@@_xdots_color_tl`).

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

```

4871 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4872 {
4873   \@@_draw_unstandard_dotted_line:nooo
4874   { #1 }
4875   \l_@@_xdots_up_tl
4876   \l_@@_xdots_down_tl
4877   \l_@@_xdots_middle_tl
4878 }
4879 \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.

```

4880 \AtBeginDocument
4881 {
4882   \IfPackageLoadedT { tikz }
4883   {
4884     \tikzset
4885     {
4886       @@_node_above / .style = { sloped , above } ,
4887       @@_node_below / .style = { sloped , below } ,
4888       @@_node_middle / .style =
4889       {

```

```

4890         sloped ,
4891         inner~sep = \c_@@_innersep_middle_dim
4892     }
4893 }
4894 }
4895 }

4896 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4897 {

```

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.

```

4898     \dim_zero_new:N \l_@@_l_dim
4899     \dim_set:Nn \l_@@_l_dim
4900     {
4901         \fp_to_dim:n
4902         {
4903             sqrt
4904             (
4905                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4906                 +
4907                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4908             )
4909         }
4910     }

```

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_unstandard_dotted_line_i:
4915     }

```

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

```

4916     \bool_if:NT \l_@@_xdots_h_labels_bool
4917     {
4918         \tikzset
4919         {
4920             @@_node_above / .style = { auto = left } ,
4921             @@_node_below / .style = { auto = right } ,
4922             @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4923         }
4924     }
4925     \tl_if_empty:nF { #4 }
4926     { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4927     \dim_zero:N \l_tmpa_dim
4928     \dim_zero:N \l_tmpb_dim
4929     \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
4930     {

```

We test whether the brace is vertical or horizontal.

```

4931         \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4932         { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
4933         { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
4934     }
4935     {

```

```

4936 \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
4937 {
4938   \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4939     { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
4940     { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
4941   }
4942 }
4943 \use:e
4944 {
4945   \exp_not:N \begin { scope }
4946     [ shift = {(\dim_use:N \l_tmpa_dim, \dim_use:N \l_tmpb_dim)} ]
4947   }
4948 \draw
4949 [ #1 ]
4950 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
4951 -- node [ @@_node_middle] { $ \scriptstyle #4 $ }
4952 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4953 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4954 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4955 \end { scope }
4956 \end { scope }
4957 }
4958 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4959 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4960 {
4961   \dim_set:Nn \l_tmpa_dim
4962   {
4963     \l_@@_x_initial_dim
4964     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4965     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4966   }
4967   \dim_set:Nn \l_tmpb_dim
4968   {
4969     \l_@@_y_initial_dim
4970     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4971     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4972   }
4973   \dim_set:Nn \l_@@_tmpc_dim
4974   {
4975     \l_@@_x_final_dim
4976     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4977     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4978   }
4979   \dim_set:Nn \l_@@_tmpd_dim
4980   {
4981     \l_@@_y_final_dim
4982     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4983     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4984   }
4985   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4986   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4987   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4988   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4989 }

```

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

```

4990 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4991 {
4992   {

```

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.

```

4993 \dim_zero_new:N \l_@@_l_dim
4994 \dim_set:Nn \l_@@_l_dim
4995 {
4996   \fp_to_dim:n
4997   {
4998     sqrt
4999     (
5000       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5001       +
5002       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5003     )
5004   }
5005 }

```

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.

```

5006 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
5007 {
5008   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
5009   \@@_draw_standard_dotted_line_i:
5010 }
5011 }
5012 \bool_lazy_all:nF
5013 {
5014   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
5015   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
5016   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
5017 }
5018 \@@_labels_standard_dotted_line:
5019 }
5020 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
5021 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
5022 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

5023 \int_set:Nn \l_tmpa_int
5024 {
5025   \dim_ratio:nn
5026   {
5027     \l_@@_l_dim
5028     - \l_@@_xdots_shorten_start_dim
5029     - \l_@@_xdots_shorten_end_dim
5030   }
5031   \l_@@_xdots_inter_dim
5032 }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

5033 \dim_set:Nn \l_tmpa_dim
5034 {
5035   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5036   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5037 }
5038 \dim_set:Nn \l_tmpb_dim
5039 {
5040   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5041   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5042 }

```

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

```

5043   \dim_gadd:Nn \l_@@_x_initial_dim
5044   {
5045     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5046     \dim_ratio:nn
5047     {
5048       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5049       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5050     }
5051     { 2 \l_@@_l_dim }
5052   }
5053   \dim_gadd:Nn \l_@@_y_initial_dim
5054   {
5055     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5056     \dim_ratio:nn
5057     {
5058       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5059       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5060     }
5061     { 2 \l_@@_l_dim }
5062   }
5063   \pgf@relevantforpicturesizefalse
5064   \int_step_inline:nnn \c_zero_int \l_tmpa_int
5065   {
5066     \pgfpathcircle
5067     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5068     \l_@@_xdots_radius_dim
5069     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
5070     \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
5071   }
5072   \pgfusepathqfill
5073 }

5074 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
5075 {
5076   \pgfscope
5077   \pgftransformshift
5078   {
5079     \pgfpointlineattime { 0.5 }
5080     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5081     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
5082   }
5083   \fp_set:Nn \l_tmpa_fp
5084   {
5085     atand
5086     (
5087       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
5088       \l_@@_x_final_dim - \l_@@_x_initial_dim
5089     )
5090   }
5091   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
5092   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
5093   \tl_if_empty:NF \l_@@_xdots_middle_tl
5094   {
5095     \begin { pgfscope }
5096     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
5097     \pgfnode
5098     { rectangle }
5099     { center }
5100     {
5101       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }

```

```

5102         {
5103             $ % $
5104             \scriptstyle \l_@@_xdots_middle_tl
5105             $ % $
5106         }
5107     }
5108     { }
5109     {
5110         \pgfsetfillcolor { white }
5111         \pgfusepath { fill }
5112     }
5113     \end { pgfscope }
5114 }
5115 \tl_if_empty:NF \l_@@_xdots_up_tl
5116 {
5117     \pgfnode
5118     { rectangle }
5119     { south }
5120     {
5121         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5122         {
5123             $ % $
5124             \scriptstyle \l_@@_xdots_up_tl
5125             $ % $
5126         }
5127     }
5128     { }
5129     { \pgfusepath { } }
5130 }
5131 \tl_if_empty:NF \l_@@_xdots_down_tl
5132 {
5133     \pgfnode
5134     { rectangle }
5135     { north }
5136     {
5137         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5138         {
5139             $ % $
5140             \scriptstyle \l_@@_xdots_down_tl
5141             $ % $
5142         }
5143     }
5144     { }
5145     { \pgfusepath { } }
5146 }
5147 \endpgfscope
5148 }

```

18 User commands available in the new environments

The commands `\@@_Ldots:`, `\@@_Cdots:`, `\@@_Vdots:`, `\@@_Ddots:` and `\@@_Iddots:` will be linked to `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots` and `\Iddots` in the environments `{NiceArray}` (the other environments of `nicematrix` rely upon `{NiceArray}`).

The syntax of these commands uses the character `_` as embellishment and that's why we have to insert a character `_` in the *arg spec* of these commands. However, we don't know the future catcode of `_` in the main document (maybe the user will use `underscore`, and, in that case, the catcode is 13 because `underscore` activates `_`). That's why these commands will be defined in a `\AtBeginDocument` and the *arg spec* will be rescanned.


```

5149 \AtBeginDocument
5150 {

```

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

```

5151 \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
5152 \cs_new_protected:Npn \@@_Ldots: { \@@_collect_options:n { \@@_Ldots_i } }
5153 \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5154 {
5155   \int_if_zero:nTF \c@jCol
5156   { \@@_error:nn { in~first~col } { \Ldots } }
5157   {
5158     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5159     { \@@_error:nn { in~last~col } { \Ldots } }
5160     {
5161       \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5162       { #1 , down = #2 , up = #3 , middle = #4 }
5163     }
5164   }
5165   \bool_if:NF \l_@@_nullify_dots_bool
5166   { \phantom { \ensuremath { \@@_old_ldots: } } }
5167   \bool_gset_true:N \g_@@_empty_cell_bool
5168 }

```

```

5169 \cs_new_protected:Npn \@@_Cdots: { \@@_collect_options:n { \@@_Cdots_i } }
5170 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5171 {
5172   \int_if_zero:nTF \c@jCol
5173   { \@@_error:nn { in~first~col } { \Cdots } }
5174   {
5175     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5176     { \@@_error:nn { in~last~col } { \Cdots } }
5177     {
5178       \@@_instruction_of_type:nnn { \c_false_bool } { \Cdots }
5179       { #1 , down = #2 , up = #3 , middle = #4 }
5180     }
5181   }
5182   \bool_if:NF \l_@@_nullify_dots_bool
5183   { \phantom { \ensuremath { \@@_old_cdots: } } }
5184   \bool_gset_true:N \g_@@_empty_cell_bool
5185 }

```

```

5186 \cs_new_protected:Npn \@@_Vdots: { \@@_collect_options:n { \@@_Vdots_i } }
5187 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5188 {
5189   \int_if_zero:nTF \c@iRow
5190   { \@@_error:nn { in~first~row } { \Vdots } }
5191   {
5192     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5193     { \@@_error:nn { in~last~row } { \Vdots } }
5194     {
5195       \@@_instruction_of_type:nnn { \c_false_bool } { \Vdots }
5196       { #1 , down = #2 , up = #3 , middle = #4 }
5197     }
5198   }
5199   \bool_if:NF \l_@@_nullify_dots_bool
5200   { \phantom { \ensuremath { \@@_old_vdots: } } }
5201   \bool_gset_true:N \g_@@_empty_cell_bool
5202 }

```

```

5203 \cs_new_protected:Npn \@@_Ddots: { \@@_collect_options:n { \@@_Ddots_i } }

```

```

5204 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5205 {
5206   \int_case:nnF \c@iRow
5207   {
5208     0 { \@@_error:nn { in-first~row } { \Ddots } }
5209     \l_@@_last_row_int { \@@_error:nn { in-last~row } { \Ddots } }
5210   }
5211   {
5212     \int_case:nnF \c@jCol
5213     {
5214       0 { \@@_error:nn { in-first~col } { \Ddots } }
5215       \l_@@_last_col_int { \@@_error:nn { in-last~col } { \Ddots } }
5216     }
5217     {
5218       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5219       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5220       { #1 , down = #2 , up = #3 , middle = #4 }
5221     }
5222   }
5223 }
5224 \bool_if:NF \l_@@_nullify_dots_bool
5225 { \phantom { \ensuremath { \@@_old_ddots: } } }
5226 \bool_gset_true:N \g_@@_empty_cell_bool
5227 }

5228 \cs_new_protected:Npn \@@_Iddots:
5229 { \@@_collect_options:n { \@@_Iddots_i } }
5230 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5231 {
5232   \int_case:nnF \c@iRow
5233   {
5234     0 { \@@_error:nn { in-first~row } { \Iddots } }
5235     \l_@@_last_row_int { \@@_error:nn { in-last~row } { \Iddots } }
5236   }
5237   {
5238     \int_case:nnF \c@jCol
5239     {
5240       0 { \@@_error:nn { in-first~col } { \Iddots } }
5241       \l_@@_last_col_int { \@@_error:nn { in-last~col } { \Iddots } }
5242     }
5243     {
5244       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5245       \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { Iddots }
5246       { #1 , down = #2 , up = #3 , middle = #4 }
5247     }
5248   }
5249   \bool_if:NF \l_@@_nullify_dots_bool
5250   { \phantom { \ensuremath { \@@_old_iddots: } } }
5251   \bool_gset_true:N \g_@@_empty_cell_bool
5252 }
5253 }

```

End of the \AddToHook.

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

```

5254 \keys_define:nn { nicematrix / Ddots }
5255 {
5256   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5257   draw-first .value_forbidden:n = true ,
5258 }

```

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

```

5259 \cs_new_protected:Npn \@@_Hspace:
5260 {
5261   \bool_gset_true:N \g_@@_empty_cell_bool
5262   \hspace
5263 }

```

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`.

```

5264 \cs_new_eq:NN \@@_old_multicolumn: \multicolumn

```

The command `\@@_Hdotsfor` will be linked to `\Hdotsfor` in `{NiceArrayWithDelims}`. TikZ nodes are created also in the implicit cells of the `\Hdotsfor` (maybe we should modify that point).

This command must *not* be protected since it begins with `\multicolumn`.

```

5265 \cs_new:Npn \@@_Hdotsfor:
5266 {
5267   \bool_lazy_and:nnTF
5268   { \int_if_zero_p:n \c@jCol }
5269   { \int_if_zero_p:n \l_@@_first_col_int }
5270   {
5271     \bool_if:NTF \g_@@_after_col_zero_bool
5272     {
5273       \multicolumn { 1 } { c } { }
5274       \@@_Hdotsfor_i:
5275     }
5276     { \@@_fatal:n { Hdotsfor~in~col~0 } }
5277   }
5278   {
5279     \multicolumn { 1 } { c } { }
5280     \@@_Hdotsfor_i:
5281   }
5282 }

```

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:`).

```

5283 \AtBeginDocument
5284 {

```

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

```

5285   \cs_new_protected:Npn \@@_Hdotsfor_i:
5286   { \@@_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`).

```

5287   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m O { } E { _ ^ : } { { } { } { } } }
5288   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5289   {
5290     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5291     {
5292       \@@_Hdotsfor:nnnn
5293       { \int_use:N \c@iRow }
5294       { \int_use:N \c@jCol }
5295       { #2 }
5296       {
5297         #1 , #3 ,
5298         down = \exp_not:n { #4 } ,
5299         up = \exp_not:n { #5 } ,
5300         middle = \exp_not:n { #6 }
5301       }
5302     }
5303     \prg_replicate:nn { #2 - 1 }

```

```

5304         {
5305             &
5306             \multicolumn { 1 } { c } { }
5307             \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5308         }
5309     }
5310 }

```

```

5311 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5312 {
5313     \bool_set_false:N \l_@@_initial_open_bool
5314     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5315     \int_set:Nn \l_@@_initial_i_int { #1 }
5316     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5317     \int_compare:nNnTF { #2 } = 1
5318     {
5319         \int_set:Nn \l_@@_initial_j_int 1
5320         \bool_set_true:N \l_@@_initial_open_bool
5321     }
5322     {
5323         \cs_if_exist:cTF
5324         {
5325             pgf @ sh @ ns @ \@@_env:
5326             - \int_use:N \l_@@_initial_i_int
5327             - \int_eval:n { #2 - 1 }
5328         }
5329         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5330         {
5331             \int_set:Nn \l_@@_initial_j_int { #2 }
5332             \bool_set_true:N \l_@@_initial_open_bool
5333         }
5334     }
5335     \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5336     {
5337         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5338         \bool_set_true:N \l_@@_final_open_bool
5339     }
5340     {
5341         \cs_if_exist:cTF
5342         {
5343             pgf @ sh @ ns @ \@@_env:
5344             - \int_use:N \l_@@_final_i_int
5345             - \int_eval:n { #2 + #3 }
5346         }
5347         { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5348         {
5349             \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5350             \bool_set_true:N \l_@@_final_open_bool
5351         }
5352     }
5353     \bool_if:NT \g_@@_aux_found_bool
5354     {
5355         {
5356             \@@_open_shorten:
5357             \int_if_zero:nTF { #1 }
5358             { \color { nicematrix-first-row } }
5359             {
5360                 \int_compare:nNnT { #1 } = \g_@@_row_total_int
5361                 { \color { nicematrix-last-row } }
5362             }

```

```

5363         \keys_set:nn { nicematrix / xdots } { #4 }
5364         \@@_color:o \l_@@_xdots_color_tl
5365         \@@_actually_draw_Ldots:
5366     }
5367 }

```

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

```

5368     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5369     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - #1 } { } }
5370 }

```

```

5371 \AtBeginDocument
5372 {
5373     \cs_new_protected:Npn \@@_Vdotsfor:
5374     { \@@_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`).

```

5375     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m O { } E { _ ^ : } { { } { } { } } }
5376     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5377     {
5378         \bool_gset_true:N \g_@@_empty_cell_bool
5379         \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5380         {
5381             \@@_Vdotsfor:nnnn
5382             { \int_use:N \c@iRow }
5383             { \int_use:N \c@jCol }
5384             { #2 }
5385             {
5386                 #1 , #3 ,
5387                 down = \exp_not:n { #4 } ,
5388                 up = \exp_not:n { #5 } ,
5389                 middle = \exp_not:n { #6 }
5390             }
5391         }
5392     }
5393 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5394 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5395 {
5396     \bool_set_false:N \l_@@_initial_open_bool
5397     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

5398     \int_set:Nn \l_@@_initial_j_int { #2 }
5399     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5400     \int_compare:nNnTF { #1 } = 1
5401     {
5402         \int_set:Nn \l_@@_initial_i_int 1
5403         \bool_set_true:N \l_@@_initial_open_bool
5404     }
5405     {
5406         \cs_if_exist:cTF
5407         {
5408             pgf @ sh @ ns @ \@@_env:

```

```

5409         - \int_eval:n { #1 - 1 }
5410         - \int_use:N \l_@@_initial_j_int
5411     }
5412     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5413     {
5414         \int_set:Nn \l_@@_initial_i_int { #1 }
5415         \bool_set_true:N \l_@@_initial_open_bool
5416     }
5417 }
5418 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5419 {
5420     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5421     \bool_set_true:N \l_@@_final_open_bool
5422 }
5423 {
5424     \cs_if_exist:cTF
5425     {
5426         pgf @ sh @ ns @ \@@_env:
5427         - \int_eval:n { #1 + #3 }
5428         - \int_use:N \l_@@_final_j_int
5429     }
5430     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5431     {
5432         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5433         \bool_set_true:N \l_@@_final_open_bool
5434     }
5435 }
5436 \bool_if:NT \g_@@_aux_found_bool
5437 {
5438     {
5439         \@@_open_shorten:
5440         \int_if_zero:nTF { #2 }
5441         { \color { nicematrix-first-col } }
5442         {
5443             \int_compare:nNnT { #2 } = \g_@@_col_total_int
5444             { \color { nicematrix-last-col } }
5445         }
5446         \keys_set:nn { nicematrix / xdots } { #4 }
5447         \@@_color:o \l_@@_xdots_color_tl
5448         \bool_if:NTF \l_@@_Vbrace_bool
5449         \@@_actually_draw_Vbrace:
5450         \@@_actually_draw_Vdots:
5451     }
5452 }

```

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

```

5453     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5454     { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5455 }

```

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

```

5456 \NewDocumentCommand \@@_rotate: { 0 { } }
5457 {
5458     \bool_gset_true:N \g_@@_rotate_bool
5459     \keys_set:nn { nicematrix / rotate } { #1 }
5460     \ignorespaces
5461 }

```

The command `\@@_rotate_p_col:` will be linked to `\rotate` in the the cells of the columns of type `p` and *al*.

```

5462 \cs_new_protected:Npn \@@_rotate_p_col: { \@@_error:n { rotate~in~p~col } }

5463 \keys_define:nn { nicematrix / rotate }
5464 {
5465   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5466   c .value_forbidden:n = true ,
5467   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5468 }

```

19 The command `\line` accessible in `\CodeAfter`

In the `\CodeAfter`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format *i-j*) and draws a dotted line between these cells. In fact, it also works with names of blocks.

First, we write a command with the following behaviour:

- If the argument is of the format *i-j*, our command applies the command `\int_eval:n` to *i* and *j* ;
- If not (that is to say, when it's a name of a `\Block`), the argument is left unchanged.

This must *not* be protected (and is, of course fully expandable).¹⁴

```

5469 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5470 {
5471   \tl_if_empty:nTF { #2 }
5472   { #1 }
5473   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5474 }
5475 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5476 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

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

```

5477 \AtBeginDocument
5478 {

```

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

```

5479   \tl_set_rescan:Nnn \l_tmpa_tl { }
5480   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5481   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5482   {
5483     {
5484       \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5485       \@@_color:o \l_@@_xdots_color_tl
5486       \use:e
5487       {
5488         \@@_line_i:nn
5489         { \@@_double_int_eval:n #2 - \q_stop }
5490         { \@@_double_int_eval:n #3 - \q_stop }
5491       }

```

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

```

5492     }
5493   }
5494 }

5495 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5496 {
5497   \bool_set_false:N \l_@@_initial_open_bool
5498   \bool_set_false:N \l_@@_final_open_bool
5499   \bool_lazy_or:nnTF
5500     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5501     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5502     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

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

```

5503   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5504 }

5505 \AtBeginDocument
5506 {
5507   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5508   {

```

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:`.

```

5509     \c_@@_pgfortikzpicture_tl
5510     \@@_draw_line_iii:nn { #1 } { #2 }
5511     \c_@@_endpgfortikzpicture_tl
5512   }
5513 }

```

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

```

5514 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5515 {
5516   \pgfrememberpicturepositiononpagetrue
5517   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5518   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5519   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5520   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5521   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5522   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5523   \@@_draw_line:
5524 }

```

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.

```
5525 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNnT \c@iRow < }
5526 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNnF \c@jCol < }
```

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

```
5527 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5528 {
5529   \tl_gput_right:Ne \g_@@_row_style_tl
5530   {
```

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

```
5531   \exp_not:N
5532   \@@_if_row_less_than:nn
5533   { \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).

```
5534   {
5535     \exp_not:N
5536     \@@_if_col_greater_than:nn
5537     { \int_use:N \c@jCol }
5538     { \exp_not:n { #1 } \scan_stop: }
5539   }
5540 }
5541 }
5542 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
```

```
5543 \keys_define:nn { nicematrix / RowStyle }
5544 {
5545   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5546   cell-space-top-limit+ .code:n =
5547     \dim_set:Nn \l_tmpa_dim { \l_@@_cell_space_top_limit_dim + #1 } ,
5548   cell-space-top-limit+ .value_required:n = true ,
5549   cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
5550   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5551   cell-space-bottom-limit+ .code:n =
5552     \dim_set:Nn \l_tmpb_dim { \l_@@_cell_space_bottom_limit_dim + #1 } ,
5553   cell-space-bottom-limit+ .value_required:n = true ,
5554   cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
5555   cell-space-limits .meta:n =
5556     {
5557       cell-space-top-limit = #1 ,
5558       cell-space-bottom-limit = #1 ,
5559     } ,
5560   cell-space-limits+ .meta:n =
5561     {
5562       cell-space-top-limit += #1 ,
5563       cell-space-bottom-limit += #1 ,
5564     } ,
5565   cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
5566   color .tl_set:N = \l_@@_color_tl ,
5567   color .value_required:n = true ,
5568   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5569   nb-rows .code:n =
5570     \str_if_eq:eeTF { #1 } { * }
5571     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5572     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5573   nb-rows .value_required:n = true ,
5574   fill .tl_set:N = \l_@@_fill_tl ,
5575   fill .value_required:n = true ,
```

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

```

5576     opacity .tl_set:N = \l_@@_opacity_tl ,
5577     opacity .value_required:n = true ,
5578     rowcolor .tl_set:N = \l_@@_fill_tl ,
5579     rowcolor .value_required:n = true ,
5580     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5581     rounded-corners .default:n = 4 pt ,
5582     unknown .code:n =
5583       \@@_unknown_key:nn
5584       { nicematrix / RowStyle }
5585       { Unknown-key-for-RowStyle }
5586   }

```

```

5587 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5588 {
5589   \group_begin:
5590   \tl_clear:N \l_@@_fill_tl
5591   \tl_clear:N \l_@@_opacity_tl
5592   \tl_clear:N \l_@@_color_tl
5593   \int_set:Nn \l_@@_key_nb_rows_int 1
5594   \dim_zero:N \l_@@_rounded_corners_dim
5595   \dim_zero:N \l_tmpa_dim
5596   \dim_zero:N \l_tmpb_dim
5597   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5598   \tl_if_empty:NF \l_@@_fill_tl
5599   {
5600     \@@_add_opacity_to_fill:
5601     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5602     {

```

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

```

5603       \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5604       { \int_use:N \c@iRow - \int_use:N \c@jCol }
5605       {
5606         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5607         - *
5608       }
5609       { \dim_use:N \l_@@_rounded_corners_dim }
5610     }
5611   }
5612   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5613   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5614   {
5615     \@@_put_in_row_style:e
5616     {
5617       \@@_put_in_cell_after_hook:n
5618       {

```

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

```

5619       \dim_set:Nn \l_@@_cell_space_top_limit_dim
5620       { \dim_use:N \l_tmpa_dim }
5621     }
5622   }
5623 }

```

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

```

5624   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5625   {
5626     \@@_put_in_row_style:e
5627     {

```

```

5628         \@@_put_in_cell_after_hook:n
5629         {
5630             \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5631             { \dim_use:N \l_tmpb_dim }
5632         }
5633     }
5634 }

```

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

```

5635     \tl_if_empty:NF \l_@@_color_tl
5636     {
5637         \@@_put_in_row_style:e
5638         {
5639             \mode_leave_vertical:
5640             \@@_color:n { \l_@@_color_tl }
5641         }
5642     }

```

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

```

5643     \bool_if:NT \l_@@_bold_row_style_bool
5644     {
5645         \@@_put_in_row_style:n
5646         {
5647             \exp_not:n
5648             {
5649                 \if_mode_math:
5650                 $ % $
5651                 \bfseries \boldmath
5652                 $ % $
5653             \else:
5654             \bfseries \boldmath
5655             \fi:
5656         }
5657     }
5658 }
5659 \group_end:
5660 \g_@@_row_style_tl
5661 \ignorespaces
5662 }

```

The following commande must *not* be protected.

```

5663 \cs_new:Npn \@@_rounded_from_row:n #1
5664 {
5665     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

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

```

5666     { \int_eval:n { #1 } - 1 }
5667     {
5668         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5669         - \exp_not:n { \int_use:N \c@jCol }
5670     }
5671     { \dim_use:N \l_@@_rounded_corners_dim }
5672 }

```

21 Colors of cells, rows and columns

We want to avoid the thin white lines that are shown in some PDF viewers (eg: with the engine MuPDF used by SumatraPDF). That’s why we try to draw rectangles of the same color in the same instruction `\pgfusepath { fill }` (and they will be in the same instruction `fill`—coded `f`—in the resulting PDF).

The commands `\@@_rowcolor`, `\@@_columncolor`, `\@@_rectanglecolor` and `\@@_rowlistcolors` don't directly draw the corresponding rectangles. Instead, they store their instructions color by color:

- A sequence `\g_@@_colors_seq` will be built containing all the colors used by at least one of these instructions. Each *color* may be prefixed by its color model (eg: `[gray]{0.5}`).
- For the color whose index in `\g_@@_colors_seq` is equal to *i*, a list of instructions which use that color will be constructed in the token list `\g_@@_color_i_tl`. In that token list, the instructions will be written using `\@@_cartesian_color:nn` and `\@@_rectanglecolor:nn`.

#1 is the color and #2 is an instruction using that color. Despite its name, the command `\@@_add_to_colors_seq:nn` doesn't only add a color to `\g_@@_colors_seq`: it also updates the corresponding token list `\g_@@_color_i_tl`. We add in a global way because the final user may use the instructions such as `\cellcolor` in a loop of `pgffor` in the `\CodeBefore` (and we recall that a loop of `pgffor` is encapsulated in a group).

```
5673 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5674 {
```

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.

```
5675 \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`.

```
5676 \str_if_in:nnF { #1 } { !! }
5677 {
5678 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

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

```
5679 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5680 }
5681 \int_if_zero:nTF \l_tmpa_int
```

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

```
5682 {
5683 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5684 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5685 }
```

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

```
5686 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5687 }
5688 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
```

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

```
5689 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5690 {
5691 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5692 {
```

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

```
5693 \group_begin:
5694 \pgfsetcornersarced
5695 { \pgfpoint \l_@@_tab_rounded_corners_dim \l_@@_tab_rounded_corners_dim }
```

Because we want `nicematrix` compatible with arrays constructed by `array`, the nodes for the rows and columns (that is to say the nodes `row-i` and `col-j`) have not always the expected position, that is to say, there is sometimes a slight shifting of something such as `\arrayrulewidth`. Now, for the clipping, we have to change slightly the position of that clipping whether a rounded rectangle around the array is required. That's the point which is tested in the following line.

```
5696 \bool_if:NTF \l_@@_hvlines_bool
5697 {
```

```

5698     \pgfpathrectanglecorners
5699     {
5700         \pgfpointadd
5701         { \@@_qpoint:n { row-1 } }
5702         { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
5703     }
5704     {
5705         \pgfpointadd
5706         {
5707             \@@_qpoint:n
5708             { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
5709         }
5710         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5711     }
5712 }
5713 {
5714     \pgfpathrectanglecorners
5715     { \@@_qpoint:n { row-1 } }
5716     {
5717         \pgfpointadd
5718         {
5719             \@@_qpoint:n
5720             { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
5721         }
5722         { \pgfpoint \c_zero_dim \arrayrulewidth }
5723     }
5724 }
5725 \pgfusepath { clip }
5726 \group_end:

```

The TeX group was for \pgfsetcornersarced.

```

5727     }
5728 }

```

The macro \@@_actually_color: will actually fill all the rectangles, color by color (using the sequence \l_@@_colors_seq and all the token lists of the form \l_@@_color_i_tl).

```

5729 \cs_new_protected:Npn \@@_actually_color:
5730 {
5731     \pgfpicture
5732     \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.

```

5733     \@@_clip_with_rounded_corners:
5734     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5735     {
5736         \int_compare:nNnTF { ##1 } = 1
5737         {
5738             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5739             \use:c { g_@@_color _ 1 _tl }
5740             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5741         }
5742         {
5743             \begin { pgfscope }
5744                 \@@_color_opacity: ##2
5745                 \use:c { g_@@_color _ ##1 _tl }
5746                 \tl_gclear:c { g_@@_color _ ##1 _tl }
5747                 \pgfusepath { fill }
5748             \end { pgfscope }
5749         }
5750     }
5751 \endpgfpicture
5752 }

```

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

```

5753 \cs_new_protected:Npn \@@_color_opacity:
5754 {
5755   \peek_meaning:NTF [
5756     \@@_color_opacity:w
5757     { \@@_color_opacity:w [ ] }
5758   }

```

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.

```

5759 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5760 {
5761   \tl_clear:N \l_tmpa_tl
5762   \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.

```

5763   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5764   \tl_if_empty:NTF \l_tmpb_tl
5765     \@declaredcolor
5766     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5767 }

```

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

```

5768 \keys_define:nn { nicematrix / color-opacity }
5769 {
5770   opacity .tl_set:N          = \l_tmpa_tl ,
5771   opacity .value_required:n = true
5772 }

```

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

```

5773 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5774 {
5775   \def \l_@@_rows_tl { #1 }
5776   \def \l_@@_cols_tl { #2 }
5777   \@@_cartesian_path:
5778 }

```

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

```

5779 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5780 {
5781   \tl_if_blank:nF { #2 }
5782   {
5783     \@@_add_to_colors_seq:en
5784     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5785     { \@@_cartesian_color:nn { #3 } { - } }
5786   }
5787 }

```

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

```

5788 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5789 {
5790   \tl_if_blank:nF { #2 }
5791   {
5792     \@@_add_to_colors_seq:en
5793     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5794     { \@@_cartesian_color:nn { - } { #3 } }
5795   }
5796 }

```

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

```

5797 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5798 {
5799   \tl_if_blank:nF { #2 }
5800   {
5801     \@@_add_to_colors_seq:en
5802     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5803     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5804   }
5805 }

```

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

```

5806 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5807 {
5808   \tl_if_blank:nF { #2 }
5809   {
5810     \@@_add_to_colors_seq:en
5811     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5812     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5813   }
5814 }

```

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

```

5815 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5816 {
5817   \@@_cut_on_hyphen:w #1 \q_stop
5818   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5819   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5820   \@@_cut_on_hyphen:w #2 \q_stop
5821   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5822   \tl_set:Nx \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`.

```

5823   \@@_cartesian_path:n { #3 }
5824 }

```

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

```

5825 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5826 {
5827   \clist_map_inline:nn { #3 }
5828   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5829 }

```

```

5830 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5831 {
5832   \int_step_inline:nn \c@iRow
5833   {
5834     \int_step_inline:nn \c@jCol
5835     {
5836       \int_if_even:nTF { #####1 + ##1 }
5837       { \@@_cellcolor [ #1 ] { #2 } }
5838       { \@@_cellcolor [ #1 ] { #3 } }
5839       { ##1 - #####1 }
5840     }
5841   }
5842 }

```

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”.

```

5843 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5844 {
5845   \@@_rectanglecolor [ #1 ] { #2 }
5846   { 1 - 1 }
5847   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5848 }

5849 \keys_define:nn { nicematrix / rowcolors }
5850 {
5851   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5852   cols .tl_set:N = \l_@@_cols_tl ,
5853   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5854   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5855 }

```

The command `\rowcolors` (accessible in the `\CodeBefore`) is inspired by the command `\rowcolors` of the package `xcolor` (with the option `table`). However, the command `\rowcolors` of `nicematrix` has *not* the optional argument of the command `\rowcolors` of `xcolor`.

Here is an example: `\rowcolors{1}{blue!10}{}[respect-blocks]`.

In `nicematrix`, the command `\@@_rowcolors` appears as a special case of `\@@_rowlistcolors`.

#1 (optional) is the color space; **#2** is a list of intervals of rows; **#3** is the list of colors; **#4** is for the optional list of pairs *key=value*.

```

5856 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5857 {

```

`\l_@@_colors_seq` will be the list of colors.

```

5858   \pgfpicture
5859   \pgf@relevantforpicturesizefalse
5860   \seq_clear_new:N \l_@@_colors_seq
5861   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5862   \tl_clear_new:N \l_@@_cols_tl
5863   \tl_set:Nn \l_@@_cols_tl { - }
5864   \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).

```

5865   \int_zero_new:N \l_@@_color_int
5866   \int_set:Nn \l_@@_color_int 1
5867   \bool_if:NT \l_@@_respect_blocks_bool
5868   {

```

We don’t want to take into account a block which is entirely in the “first column” (number 0) or in the “last column” and that’s why we filter the sequence of the blocks (in the sequence `\l_tmpa_seq`).

```

5869   % modified 2026-02-18
5870   \seq_set_filter:Nnn \l_tmpa_seq \g_@@_pos_of_blocks_seq
5871   { \@@_not_in_exterior_p:nnnnn ##1 }
5872 }

```

#2 is the list of intervals of rows.

```

5873   \clist_map_inline:nn { #2 }
5874   {
5875     \tl_set:Nn \l_tmpa_tl { ##1 }
5876     \tl_if_in:NnTF \l_tmpa_tl { - }
5877     { \@@_cut_on_hyphen:w ##1 \q_stop }
5878     { \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.

```

5879     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5880     \int_set:Nn \l_@@_color_int

```



```

5881         { \bool_if:NTF \l_@@_rowcolors_restart_bool { 1 } \l_tmpa_tl }
5882     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5883     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5884     {

```

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

```

5885         \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).

```

5886         \bool_if:NT \l_@@_respect_blocks_bool
5887         {
5888             \seq_set_filter:Nn \l_tmpb_seq \l_tmpa_seq
5889             { \@@_intersect_our_row_p:nnnnn ###1 }
5890             \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

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

```

5891         }
5892         \tl_set:Ne \l_@@_rows_tl
5893         { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

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

```

5894         \tl_set:Ne \l_@@_color_tl
5895         {
5896             \@@_color_index:n
5897             {
5898                 \int_mod:nn
5899                 { \l_@@_color_int - 1 }
5900                 { \seq_count:N \l_@@_colors_seq }
5901                 + 1
5902             }
5903         }
5904         \tl_if_empty:NF \l_@@_color_tl
5905         {
5906             \use:e
5907             {
5908                 \@@_add_to_colors_seq:nn
5909                 { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5910                 { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5911             }
5912         }
5913         \int_incr:N \l_@@_color_int
5914         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5915     }
5916 }
5917 \endpgfpicture
5918 }

```

The command `\@@_color_index:n` peeks in `\l_@@_colors_seq` the color at the index `#1`. However, if that color is the symbol `=`, the previous one is poken. This macro is recursive.

```

5919 \cs_new:Npn \@@_color_index:n #1
5920 {

```

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

```

5921     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5922     { \@@_color_index:n { #1 - 1 } }
5923     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5924 }

```

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

```

5925 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5926 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5927 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5928 {
5929   \int_compare:nNt { #3 } > \l_tmpb_int
5930   { \int_set:Nn \l_tmpb_int { #3 } }
5931 }

5932 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5933 {
5934   \int_if_zero:nTF { #4 }
5935   \prg_return_false:
5936   {
5937     \int_compare:nNtTF { #2 } > \c@jCol
5938     \prg_return_false:
5939     \prg_return_true:
5940   }
5941 }

```

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

```

5942 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5943 {
5944   \int_compare:nNtTF { #1 } > \l_tmpa_int
5945   \prg_return_false:
5946   {
5947     \int_compare:nNtTF \l_tmpa_int > { #3 }
5948     \prg_return_false:
5949     \prg_return_true:
5950   }
5951 }

```

The following command uses two implicit arguments: \l_@@_rows_tl and \l_@@_cols_tl which are specifications for a set of rows and a set of columns. It creates a path but does *not* fill it. It must be filled by another command after. The argument is the radius of the corners. We define below a command \@@_cartesian_path: which corresponds to a value 0 pt for the radius of the corners. This command is, in particular, used in \@@_rectanglecolor:nnn (used in \@@_rectanglecolor, itself used in \@@_cellcolor).

```

5952 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5953 {
5954   \dim_compare:nNtTF { #1 } = \c_zero_dim
5955   {
5956     \bool_if:NTF \l_@@_nocolor_used_bool
5957     { \@@_cartesian_path_normal_ii: }
5958     {
5959       \clist_if_empty:NTF \l_@@_corners_cells_clist
5960       { \@@_cartesian_path_normal_i:n { #1 } }
5961       { \@@_cartesian_path_normal_ii: }
5962     }
5963   }
5964   { \@@_cartesian_path_normal_i:n { #1 } }
5965 }

```

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.

```

5966 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5967 {
5968   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5969 \clist_map_inline:Nn \l_@@_cols_tl
5970 {

```

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

```

5971 \def \l_tmpa_tl { ##1 }
5972 \tl_if_in:NnTF \l_tmpa_tl { - }
5973 { \@@_cut_on_hyphen:w ##1 \q_stop }
5974 { \def \l_tmpb_tl { ##1 } }
5975 \tl_if_empty:NNTF \l_tmpa_tl
5976 { \def \l_tmpa_tl { 1 } }
5977 {
5978 \str_if_eq:eeT \l_tmpa_tl { * }
5979 { \def \l_tmpa_tl { 1 } }
5980 }
5981 \int_compare:nNnT \l_tmpa_tl > \g_@@_col_total_int
5982 { \@@_error:n { Invalid~col~number } }
5983 \tl_if_empty:NNTF \l_tmpb_tl
5984 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5985 {
5986 \str_if_eq:eeT \l_tmpb_tl { * }
5987 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5988 }
5989 \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5990 { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

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

```

5991 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5992 \@@_qpoint:n { col - \l_tmpa_tl }
5993 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5994 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5995 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5996 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5997 \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.

```

5998 \clist_map_inline:Nn \l_@@_rows_tl
5999 {
6000 \def \l_tmpa_tl { #####1 }
6001 \tl_if_in:NnTF \l_tmpa_tl { - }
6002 { \@@_cut_on_hyphen:w #####1 \q_stop }
6003 { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
6004 \tl_if_empty:NNTF \l_tmpa_tl
6005 { \def \l_tmpa_tl { 1 } }
6006 {
6007 \str_if_eq:eeT \l_tmpa_tl { * }
6008 { \def \l_tmpa_tl { 1 } }
6009 }
6010 \tl_if_empty:NNTF \l_tmpb_tl
6011 { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6012 {
6013 \str_if_eq:eeT \l_tmpb_tl { * }
6014 { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6015 }
6016 \int_compare:nNnT \l_tmpa_tl > \g_@@_row_total_int
6017 { \@@_error:n { Invalid~row~number } }
6018 \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
6019 { \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`.

```

6020 \cs_if_exist:cF
6021 { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
6022 {
6023 \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
6024 \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6025 \@@_qpoint:n { row - \l_tmpa_tl }
6026 \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6027 \pgfpathrectanglecorners
6028 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }

```

```

6029         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6030     }
6031 }
6032 }
6033 }

```

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

```

6034 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
6035 {
6036     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6037     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6038     \clist_map_inline:Nn \l_@@_cols_tl
6039     {
6040         \@@_qpoint:n { col - ##1 }
6041         \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
6042         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6043         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6044         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
6045         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

6046     \clist_map_inline:Nn \l_@@_rows_tl
6047     {
6048         \@@_if_in_corner:nF { #####1 - ##1 }
6049         {
6050             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
6051             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6052             \@@_qpoint:n { row - #####1 }
6053             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6054             \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
6055             {
6056                 \pgfpathrectanglecorners
6057                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6058                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6059             }
6060         }
6061     }
6062 }
6063 }

```

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`).

```

6064 \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

```

6065 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
6066 {
6067     \bool_set_true:N \l_@@_nocolor_used_bool
6068     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6069     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6070     \clist_map_inline:Nn \l_@@_rows_tl
6071     {
6072         \clist_map_inline:Nn \l_@@_cols_tl
6073         { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
6074     }
6075 }

```

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.

```
6076 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
6077 {
6078   \clist_set_eq:NN \l_tmpa_clist #1
6079   \clist_clear:N #1
6080   \clist_map_inline:Nn \l_tmpa_clist
6081   {
```

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

```
6082   \def \l_tmpa_tl { ##1 }
6083   \tl_if_in:NnTF \l_tmpa_tl { - }
6084   { \@@_cut_on_hyphen:w ##1 \q_stop }
6085   { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
6086   \bool_lazy_or:nnT
6087   { \str_if_eq_p:ee \l_tmpa_tl { * } }
6088   { \tl_if_blank_p:o \l_tmpa_tl }
6089   { \def \l_tmpa_tl { 1 } }
6090   \bool_lazy_or:nnT
6091   { \str_if_eq_p:ee \l_tmpb_tl { * } }
6092   { \tl_if_blank_p:o \l_tmpb_tl }
6093   { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6094   \int_compare:nNnT \l_tmpb_tl > { #2 }
6095   { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6096   \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
6097   { \clist_put_right:Nn #1 { #####1 } }
6098 }
6099 }
```

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

```
6100 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
6101 {
6102   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6103   {
```

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`).

```
6104   \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
6105   { \int_use:N \c@iRow - \int_use:N \c@jCol }
6106 }
6107 \ignorespaces
6108 }

6109 \NewDocumentCommand \@@_cellcolor_error { 0 { } m }
6110 { \@@_error:n { cellcolor~in~Block } }
6111 % \end{macrocode}
6112 %
6113 % \begin{macrocode}
6114 \NewDocumentCommand \@@_rowcolor_error { 0 { } m }
6115 { \@@_error:n { rowcolor~in~Block } }
6116 % \end{macrocode}
6117 %
6118 % \bigskip
6119 % The following command will be linked to |\rowcolor| in the tabular.
6120 % \begin{macrocode}
6121 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
6122 {
6123   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6124   {
6125     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
6126     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6127     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
6128   }

```

```

6129     \ignorespaces
6130 }

```

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

```

6131 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
6132 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around #2 and #3 are mandatory.

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

```

6133 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
6134 {

```

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`.

```

6135     \seq_gclear:N \g_tmpa_seq
6136     \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6137     { \@@_rowlistcolors_tabular:nnnn #1 }
6138     \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`.

```

6139     \seq_gput_right:Ne \g_@@_rowlistcolors_seq
6140     {
6141         { \int_use:N \c@iRow }
6142         { \exp_not:n { #1 } }
6143         { \exp_not:n { #2 } }
6144         { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
6145     }
6146     \ignorespaces
6147 }

```

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`).

```

6148 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
6149 {
6150     \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).

```

6151     { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
6152     {
6153         \tl_gput_right:Ne \g_@@_pre_code_before_tl
6154         {
6155             \@@_rowlistcolors
6156             [ \exp_not:n { #2 } ]
6157             { #1 - \int_eval:n { \c@iRow - 1 } }
6158             { \exp_not:n { #3 } }
6159             [ \exp_not:n { #4 } ]
6160         }
6161     }
6162 }

```

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.

```

6163 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
6164 {
6165   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6166     { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
6167   \seq_gclear:N \g_@@_rowlistcolors_seq
6168 }

6169 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6170 {
6171   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6172     { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6173 }

```

The first mandatory argument of the command `\@@_rowlistcolors` which is writtent in the pre-`\CodeBefore` is of the form `i`: it means that the command must be applied to all the rows from the row `i` until the end of the tabular.

```

6174 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
6175 {

```

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).

```

6176   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
6177   {

```

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).

```

6178     \tl_gput_left:Ne \g_@@_pre_code_before_tl
6179     {
6180       \exp_not:N \columncolor [ #1 ]
6181       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6182     }
6183   }
6184 }

6185 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6186 {
6187   \clist_map_inline:nn { #1 }
6188   {
6189     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6190       { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6191     \columncolor { nocolor } { ##1 }
6192   }
6193 }

6194 \cs_new_protected:Npn \@@_EmptyRow:n #1
6195 {
6196   \clist_map_inline:nn { #1 }
6197   {
6198     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6199       { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6200     \rowcolor { nocolor } { ##1 }
6201   }
6202 }

```

22 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnntype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```
6203 \cs_new_eq:NN \OnlyMainNiceMatrix \use:n
```

Another definition of `\OnlyMainNiceMatrix` will be linked to the command in the environments of `nicematrix`. Here is that definition, called `\@@_OnlyMainNiceMatrix:n`.

```
6204 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6205 {
6206   \int_if_zero:nTF \l_@@_first_col_int
6207   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6208   {
6209     \int_if_zero:nTF \c@jCol
6210     {
6211       \int_compare:nNnF \c@iRow = { -1 }
6212       {
6213         \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 }
6214         { #1 }
6215       }
6216     }
6217     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6218   }
6219 }
```

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.

```
6220 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6221 {
6222   \int_if_zero:nF \c@iRow
6223   {
6224     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6225     {
6226       \int_compare:nNnT \c@jCol > \c_zero_int
6227       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6228     }
6229   }
6230 }
```

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`.

```
6231 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6232 {
6233   \IfPackageLoadedTF { tikz }
6234   {
6235     \IfPackageLoadedTF { booktabs }
6236     { #2 }
6237     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6238   }
```



```

6239     { \@@_error:nn { TopRule~without~tikz } { #1 } }
6240 }
6241 \NewExpandableDocumentCommand { \@@_TopRule } { }
6242 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
6243 \cs_new:Npn \@@_TopRule_i:
6244 {
6245   \noalign \bgroup
6246     \peek_meaning:NTF [
6247       { \@@_TopRule_ii: }
6248       { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6249   }
6250 \NewDocumentCommand \@@_TopRule_ii: { o }
6251 {
6252   \tl_gput_right:Ne \g_@@_rules_tl
6253   {
6254     \@@_draw_hrule:n
6255     {
6256       position = \int_eval:n { \c@iRow + 1 } ,
6257       tikz =
6258       {
6259         line~width = #1 ,
6260         yshift = 0.25 \arrayrulewidth ,
6261         shorten~< = - 0.5 \arrayrulewidth
6262       } ,
6263       total~width = #1
6264     }
6265   }
6266   \skip_vertical:n { \belowrulesep + #1 }
6267   \egroup
6268 }
6269 \NewExpandableDocumentCommand { \@@_BottomRule } { }
6270 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }
6271 \cs_new:Npn \@@_BottomRule_i:
6272 {
6273   \noalign \bgroup
6274     \peek_meaning:NTF [
6275       { \@@_BottomRule_ii: }
6276       { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6277   }
6278 \NewDocumentCommand \@@_BottomRule_ii: { o }
6279 {
6280   \tl_gput_right:Ne \g_@@_rules_tl
6281   {
6282     \@@_draw_hrule:n
6283     {
6284       position = \int_eval:n { \c@iRow + 1 } ,
6285       tikz =
6286       {
6287         line~width = #1 ,
6288         yshift = 0.25 \arrayrulewidth ,
6289         shorten~< = - 0.5 \arrayrulewidth
6290       } ,
6291       total~width = #1 ,
6292     }
6293   }
6294   \skip_vertical:N \aboverulesep
6295   \@@_create_row_node_i:
6296   \skip_vertical:n { #1 }
6297   \egroup
6298 }
6299 \NewExpandableDocumentCommand { \@@_MidRule } { }

```

```

6300 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6301 \cs_new:Npn \@@_MidRule_i:
6302 {
6303   \noalign \bgroup
6304     \peek_meaning:NTF [
6305       { \@@_MidRule_ii: }
6306       { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6307   }
6308   \NewDocumentCommand \@@_MidRule_ii: { o }
6309   {
6310     \skip_vertical:N \aboverulesep
6311     \@@_create_row_node_i:
6312     \tl_gput_right:Ne \g_@@_rules_tl
6313     {
6314       \@@_draw_hrulerule:n
6315       {
6316         position = \int_eval:n { \c@iRow + 1 } ,
6317         tikz =
6318         {
6319           line-width = #1 ,
6320           yshift = 0.25 \arrayrulewidth ,
6321           shorten- < = - 0.5 \arrayrulewidth
6322         } ,
6323         total-width = #1 ,
6324       }
6325     }
6326     \skip_vertical:n { \belowrulesep + #1 }
6327     \egroup
6328   }

```

General system for drawing rules

```

6329 \AtBeginDocument
6330 {
6331   \cs_new_protected:Npe \@@_draw_rules:
6332   {
6333     \c_@@_pgfortikzpicture_tl
6334     \@@_draw_rules_i:
6335     \c_@@_endpgfortikzpicture_tl
6336   }
6337 }
6338 \cs_new_protected:Npn \@@_draw_rules_i:
6339 {
6340   \pgfrememberpicturepositiononpagetrue
6341   \pgf@relevantforpicturesizefalse
6342   \pgfsetrectcap
6343   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6344   \CT@arc@
6345   \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_key_hlines:
6346   \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_key_vlines:
6347   \g_@@_rules_tl
6348 }

```

When a command, environment or “subsystem” of `nicematrix` wants to draw a rule, it will write in `\g_@@_rules_tl` a command `\@@_draw_vrule:n` or `\@@_draw_hrulerule: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 which is a kind of “internal set of keys”.

```

6349 \keys_define:nn { nicematrix / rules-after }
6350 {
6351   position .int_set:N = \l_@@_position_int ,
6352   start .int_set:N = \l_@@_start_int ,

```

```

6353     start .initial:n = 1 ,
6354     end .int_set:N = \l_@@_end_int ,
6355     multiplicity .code:n = \@@_set_multiplicity:n { #1 } ,
6356     dotted .code:n =
6357         \bool_set_true:N \l_@@_dotted_bool
6358         \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
6359         \socket_assign_plug:nn { nicematrix / hsegment } { dotted } ,
6360     dotted .value_forbidden: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]`.

```

6361     color .code:n =
6362         \@@_set_CTarc:n { #1 }
6363         \CT@arc@
6364         \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6365     color .value_required:n = true ,
6366     sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6367     sep-color .value_required:n = true ,

```

Example for the key `total-width`:

```

\NiceMatrixOptions
{
  custom-line =
  {
    letter = I ,
    tikz = { line width = 1pt , dotted } ,
    total-width = 1 pt
  }
}

6368     total-width .dim_set:N = \l_@@_rule_width_after_dim ,
6369     unknown .code:n =
6370         \@@_unknown_key:nn
6371         { nicematrix / rules-after }
6372         { Unknown-key-for-a-rule } ,
6373     tikz .value_required:n = true
6374 }

```

If the user uses the key `tikz`, the rule (or more precisely: the different segments since a rule may be broken by blocks or others) will be drawn with TikZ.

```

6375 \AtBeginDocument
6376 {
6377     \IfPackageLoadedTF { tikz }
6378     {
6379         \keys_define:nn { nicematrix / rules-after }
6380         {
6381             tikz .code:n =
6382                 \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 }
6383                 \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
6384                 \socket_assign_plug:nn { nicematrix / hsegment } { tikz }
6385         }
6386     }
6387     {
6388         \keys_define:nn { nicematrix / rules-after }
6389         { tikz .code:n = \@@_error:n { tikz-without-tikz } }
6390     }
6391 }

6392 \cs_new_protected:Npn \@@_set_multiplicity:n #1
6393 {

```

```

6394 \int_set:Nn \l_@@_multiplicity_int { #1 }
6395 \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
6396 \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
6397 }

```

The vertical rules

`\@@_draw_vrule:n` and `\@@_draw_vrule:` will appear in `\@@_draw_key_vlines:n` and in the token list `\g_@@_rules_tl` (or within other functions used in `\g_@@_rules_tl`) and those token lists will be executed within a PGF pseudo-environment.

The argument `#1` is a list of *key=value* pairs.

```

6398 \cs_new_protected:Npn \@@_draw_vrule:n #1
6399 {

```

The group is for the options.

```

6400 {
6401 \int_set_eq:NN \l_@@_end_int \c_iRow
6402 \keys_set:nn { nicematrix / rules-after } { #1 }

```

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).

```

6403 \int_compare:nNnT \l_@@_position_int < { \c_jCol + 2 }
6404 \@@_draw_vrule:
6405 }
6406 }

```

`\@@_draw_vrule:` is a function similar to `\@@_draw_vrule:n` but without the list of pairs (the rule will be drawn with the current values of the parameters for rules in the TeX group).

```

6407 \cs_new_protected:Npn \@@_draw_vrule:
6408 {
6409 \group_begin:

```

`\l_tmpa_tl` is the number of row and `\l_tmpb_tl` the number of column. When we have found a row corresponding to a rule to draw, we note its number in `\l_@@_tmpc_tl`.

```

6410 \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }

```

`\l_@@_x_initial_dim` will be the *x*-value of the rule to draw (used in all the plugs).

```

6411 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6412 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6413 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6414 \l_tmpa_tl
6415 {

```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```

6416 \@@_test_v:
6417 \bool_if:NTF \g_tmpa_bool
6418 {
6419 \int_if_zero:NTF \l_@@_segment_start_int

```

We keep in memory that we have a rule to draw. `\l_@@_segment_start_int` will be the starting row of the rule that we will have to draw.

```

6420 { \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl }
6421 { \socket_use:nn { nicematrix / draw-at-odd-vertex-v } }
6422 }
6423 {
6424 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6425 {
6426 \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }

```

The socket `vsegment` is defined below with its four plugs.

```

6427         \socket_use:n { nicematrix / vsegment }
6428         \int_zero:N \l_@@_segment_start_int
6429     }
6430 }
6431 }
6432 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6433 {
6434     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6435     \socket_use:n { nicematrix / vsegment }
6436 }
6437 \group_end:
6438 }

```

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 `create-blocks-in-col` 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.

```

6439 \cs_new_protected:Npn \@@_test_v:
6440 {
6441     \bool_gset_true:N \g_tmpa_bool
6442     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6443     \bool_if:NT \g_tmpa_bool
6444     {
6445         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6446         { \@@_test_vline_in_block:nnnnn ##1 }
6447         \bool_if:NT \g_tmpa_bool
6448         {
6449             \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6450             { \@@_test_vline_in_block:nnnnn ##1 }
6451             \bool_if:NT \g_tmpa_bool
6452             {
6453                 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6454                 { \@@_test_vline_in_stroken_block:nnnn ##1 }
6455             }
6456         }
6457     }
6458 }
6459 \socket_new:nn { nicematrix / draw-at-odd-vertex-v } { 0 }
6460 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-v } { active }
6461 {
6462     {
6463         \int_compare:nNnTF \l_@@_position_int > \c_jCol
6464         { \bool_set_false:N \l_tmpa_bool }
6465         {
6466             \@@_test_h:
6467             \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6468         }
6469         \int_compare:nNnTF \l_@@_position_int = 1
6470         { \bool_gset_false:N \g_tmpa_bool }
6471         {
6472             \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl - 1 } }
6473             \@@_test_h:
6474         }
6475         \bool_gset:Nn \g_tmpa_bool
6476         { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6477     }
6478     \bool_if:NT \g_tmpa_bool
6479     {
6480         \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }
6481         \socket_use:n { nicematrix / vsegment }
6482         \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl
6483     }

```

```

6484 }

6485 \cs_new_protected:Npn \@@_test_in_corner_v:
6486 {
6487   \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6488   {
6489     \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6490     { \bool_set_false:N \g_tmpa_bool }
6491   }
6492   {
6493     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6494     {
6495       \int_compare:nNnTF \l_tmpb_tl = 1
6496       { \bool_set_false:N \g_tmpa_bool }
6497       {
6498         \@@_if_in_corner:nT
6499         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6500         { \bool_set_false:N \g_tmpa_bool }
6501       }
6502     }
6503   }
6504 }

```

For the drawing of the (vertical) segments, we will use a socket with four plugs :

- a plug `standar` for the simple rules.
- a plug `multiple` for the rules similar to the standard rules of `array` and `colortbl`, that is to say with multiplicity, etc;
- a plug `dotted` for the rules with our own system of dotted rules;
- a plug `tikz` for the rules drawn with TikZ.

```

6505 \socket_new:nn { nicematrix / vsegment } { 0 }

```

In the following plug, the x -value for the line has been computed previously in `\l_@@_x_initial_dim`.

```

6506 \socket_new_plug:nnn { nicematrix / vsegment } { standard }
6507 {
6508   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6509   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6510   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6511   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6512   \pgfusepathqstroke
6513 }

6514 \socket_assign_plug:nn { nicematrix / vsegment } { standard }

6515 \socket_new_plug:nnn { nicematrix / vsegment } { multiple }
6516 {
6517   \dim_add:Nn \l_@@_x_initial_dim
6518   {
6519     - 0.5 \l_@@_rule_width_after_dim
6520     +
6521     ( \arrayrulewidth * \l_@@_multiplicity_int
6522       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6523   }
6524   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6525   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6526   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6527   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6528   \bool_lazy_and:nnT
6529   { \cs_if_exist_p:N \CT@drsc@ }
6530   { ! \tl_if_blank_p:o \CT@drsc@ }

```

```

6531 {
6532   {
6533     \CT@drsc@
6534     \dim_add:Nn \l_@@_y_initial_dim { 0.5 \arrayrulewidth }
6535     \dim_sub:Nn \l_@@_y_final_dim { 0.5 \arrayrulewidth }
6536     \dim_set:Nn \l_@@_tmpd_dim
6537       {
6538         \l_@@_x_initial_dim - ( \doublerulesep + \arrayrulewidth )
6539         * ( \l_@@_multiplicity_int - 1 )
6540       }
6541     \pgfpathrectanglecorners
6542       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6543       { \pgfpoint \l_@@_tmpd_dim \l_@@_y_final_dim }
6544     \pgfusepath { fill }
6545   }
6546 }
6547 \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6548 \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6549 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6550 {
6551   \dim_sub:Nn \l_@@_x_initial_dim { \arrayrulewidth + \doublerulesep }
6552   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6553   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6554 }
6555 \pgfusepathqstroke
6556 }

6557 \socket_new_plug:nnn { nicematrix / vsegment } { dotted }
6558 {
6559   \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6560   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6561   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6562   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6563   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6564   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y

```

The command `\@@_draw_line:` 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`

```

6565   \@@_draw_line:
6566 }

6567 \socket_new_plug:nnn { nicematrix / vsegment } { tikz }
6568 {
6569   {

```

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).

```

6570   \tl_if_empty:NF \l_@@_rule_color_tl
6571   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6572   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6573   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6574   \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6575   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }

```

The following line can't be short-cutted.

```

6576     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6577     % \exp_args:No \tikzset \l_@@_tikz_rule_tl
6578     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6579     ( \l_@@_x_initial_dim , \l_@@_y_initial_dim ) --
6580     ( \l_@@_x_initial_dim , \l_@@_y_final_dim ) ;
6581   }
6582 }

```

The command `\@@_draw_key_vlines:` draws the horizontal rules specified by the key `vlines`. These rules are not drawn in the blocks (even the virtual blocks determined by commands such as `\Cdots`) and in the corners — if the key `corners` is used).

```

6583 \cs_new_protected:Npn \@@_draw_key_vlines:
6584 {
6585   {
6586     \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
6587     \int_set:Nn \l_@@_end_int \c@iRow
6588     \int_step_inline:nnn
6589     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6590     {
6591       \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6592       \c@jCol
6593       { \int_eval:n { \c@jCol + 1 } } }
6594   }
6595   {
6596     \str_if_eq:eeF \l_@@_vlines_clist { all }
6597     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }

```

The argument of `\@@_vlines:n` is a list of key-value pairs. There is a key `position` which sets `\l_@@_position_int` but it's faster to do the affectation right now.

```

6598     {
6599       \int_set:Nn \l_@@_position_int { ##1 }
6600       \@@_draw_vrule:
6601     }
6602   }
6603 }
6604 }

```

The horizontal rules

`\@@_draw_hrrule:n` and `\@@_draw_hrrule:` will appear in `\@@_draw_key_hlines:n` and in the token rules `\g_@@_rules_tl` (or within other functions used in `\g_@@_rules_tl`) and those token lists will be executed within a PGF pseudo-environment.

The argument `#1` is a list of *key=value* pairs.

```

6605 \cs_new_protected:Npn \@@_draw_hrrule:n #1
6606 {
6607   {
6608     \int_set_eq:NN \l_@@_end_int \c@jCol
6609     \keys_set_known:nn { nicematrix / rules-after } { #1 }
6610     \@@_draw_hrrule:
6611   }
6612 }

```

`\@@_draw_hrrule:` is a function similar to `\@@_draw_hrrule:n` but whitout the list of pairs (the rule will be drawn with the current values of the parameters for rules in the TeX group).

```

6613 \cs_new_protected:Npn \@@_draw_hrrule:
6614 {
6615   \group_begin:

```


\l_tmpa_tl is the number of row and \l_tmpb_tl the number of column. When we have found a column corresponding to a rule to draw, we note its number in \l_@@_tmpc_tl.

```
6616 \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
```

\l_@@_y_initial_dim will be the y -value of the rule to draw (used in all the plugs).

```
6617 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6618 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6619 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6620 \l_tmpb_tl
6621 {
```

After the execution of \test_v:, \g_tmpa_bool will be equal to true if we have to draw that rule.

```
6622 \@@_test_h:
6623 \bool_if:NTF \g_tmpa_bool
6624 {
6625 \int_if_zero:nTF \l_@@_segment_start_int
```

We keep in memory that we have a segment to draw. \l_@@_segment_start_int will be the starting row of the rule that we will have to draw.

```
6626 { \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl }
6627 { \socket_use:n { nicematrix / draw-at-odd-vertex-h } }
6628 }
6629 {
6630 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6631 {
6632 \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }
```

The socket hsegment is defined below with its four plugs.

```
6633 \socket_use:n { nicematrix / hsegment }
6634 \int_zero:N \l_@@_segment_start_int
6635 }
6636 }
6637 }
6638 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6639 {
6640 \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6641 \socket_use:n { nicematrix / hsegment }
6642 }
6643 \group_end:
6644 }
```

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 create-blocks-in-col 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.

```
6645 \cs_new_protected:Npn \@@_test_h:
6646 {
6647 \bool_gset_true:N \g_tmpa_bool
6648 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6649 \bool_if:NT \g_tmpa_bool
6650 {
6651 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6652 { \@@_test_hline_in_block:nnnnn ##1 }
6653 \bool_if:NT \g_tmpa_bool
6654 {
6655 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6656 { \@@_test_hline_in_block:nnnnn ##1 }
6657 \bool_if:NT \g_tmpa_bool
6658 {
6659 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6660 { \@@_test_hline_in_stroken_block:nnnn ##1 }
6661 }
6662 }
6663 }
6664 }
```

```

6665 \socket_new:nn { nicematrix / draw-at-odd-vertex-h } { 0 }
6666 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-h } { active }
6667 {
6668   {
6669     \int_compare:nNnTF \l_@@_position_int > \c@iRow
6670     { \bool_set_false:N \l_tmpa_bool }
6671     {
6672       \@@_test_v:
6673       \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6674     }
6675     \int_compare:nNnTF \l_@@_position_int = 1
6676     { \bool_gset_false:N \g_tmpa_bool }
6677     {
6678       \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl - 1 } }
6679       \@@_test_v:
6680     }
6681     \bool_gset:Nn \g_tmpa_bool
6682     { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6683   }
6684   \bool_if:NT \g_tmpa_bool
6685   {
6686     \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }
6687     \socket_use:n { nicematrix / hsegment }
6688     \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl
6689   }
6690 }

6691 \cs_new_protected:Npn \@@_test_in_corner_h:
6692 {
6693   \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6694   {
6695     \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6696     { \bool_set_false:N \g_tmpa_bool }
6697   }
6698   {
6699     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6700     {
6701       \int_compare:nNnTF \l_tmpa_tl = 1
6702       { \bool_set_false:N \g_tmpa_bool }
6703       {
6704         \@@_if_in_corner:nT
6705         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6706         { \bool_set_false:N \g_tmpa_bool }
6707       }
6708     }
6709   }
6710 }

```

For the drawing of the (horizontal) segments, we will use a socket with four plugs :

- a plug **standar** for simple rules;
- a plug **multiplity** for the rules similar to the standard rules of `array` and `colortbl`, that is to say with multiplicity, etc;
- a plug **dotted** for the rules with our own system of dotted rules;
- a plug **tikz** for the rules drawn with TikZ.

```

6711 \socket_new:nn { nicematrix / hsegment } { 0 }

```

In the following plug, the y -value for the line has been computed previously in `\l_@@_y_initial_dim`.

```

6712 \socket_new_plug:nnn { nicematrix / hsegment } { standard }
6713 {
6714   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6715   \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6716   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6717   \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6718   \pgfusepathqstroke
6719 }

6720 \socket_assign_plug:nn { nicematrix / hsegment } { standard }

6721 \socket_new_plug:nnn { nicematrix / hsegment } { multiple }
6722 {
6723   \dim_add:Nn \l_@@_y_initial_dim
6724   {
6725     - 0.5 \l_@@_rule_width_after_dim
6726     +
6727     ( \arrayrulewidth * \l_@@_multiplicity_int
6728       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6729   }
6730   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6731   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6732   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6733   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6734   \bool_lazy_and:nnT
6735     { \cs_if_exist_p:N \CT@drsc@ }
6736     { ! \tl_if_blank_p:o \CT@drsc@ }
6737   {
6738     {
6739       \CT@drsc@
6740       \dim_set:Nn \l_@@_tmpd_dim
6741       {
6742         \l_@@_y_initial_dim - ( \doublerulesep + \arrayrulewidth )
6743         * ( \l_@@_multiplicity_int - 1 )
6744       }
6745       \pgfpathrectanglecorners
6746       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6747       { \pgfpoint \l_@@_x_final_dim \l_@@_tmpd_dim }
6748       \pgfusepathqfill
6749     }
6750   }
6751   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6752   \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
6753   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6754   {
6755     \dim_sub:Nn \l_@@_y_initial_dim { \arrayrulewidth + \doublerulesep }
6756     \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6757     \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
6758   }
6759   \pgfusepathqstroke
6760 }

```

Now, the case of a dotted line.

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6761 \socket_new_plug:nnn { nicematrix / hsegment } { dotted }
6762 {
6763   \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6764   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6765   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6766   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6767   \int_compare:nNnT \l_@@_segment_start_int = 1
6768   {
6769     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6770     \bool_if:NF \g_@@_delims_bool
6771     { \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.

```

6772   \tl_if_eq:NnF \g_@@_left_delim_tl (
6773     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6774   )
6775   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6776   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6777   \int_compare:nNnT \l_@@_segment_end_int = \c@jCol
6778   {
6779     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6780     \bool_if:NF \g_@@_delims_bool
6781     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6782     \tl_if_eq:NnF \g_@@_right_delim_tl )
6783     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6784   }

```

The command `\@@_draw_line:` 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`

```

6785   \@@_draw_line:
6786 }

6787 \socket_new_plug:nnn { nicematrix / hsegment } { tikz }
6788 {
6789   {

```

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).

```

6790   \tl_if_empty:NF \l_@@_rule_color_tl
6791   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6792   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6793   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6794   \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }

```

```

6795 \l_@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6796 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6797 % \exp_args:No \tikzset \l_@@_tikz_rule_tl
6798 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6799 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
6800 -- ( \l_@@_x_final_dim , \l_@@_y_initial_dim ) ;
6801 }
6802 }

```

The command `\l_@@_draw_key_hlines:` draws the horizontal rules specified by the key `hlines`. These rules are not drawn in the blocks (even the virtual blocks determined by commands such as `\Cdots`) and in the corners — if the key `corners` is used).

```

6803 \cs_new_protected:Npn \l_@@_draw_key_hlines:
6804 {
6805   {
6806     \int_set:Nn \l_@@_end_int \c@jCol
6807     \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
6808     \int_step_inline:nnn
6809       { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6810       {
6811         \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6812         \c@iRow
6813         { \int_eval:n { \c@iRow + 1 } } }
6814     }
6815     {
6816       \str_if_eq:eeF \l_@@_hlines_clist { all }
6817       { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }

```

The argument of `\l_@@_hlines:n` is a list of key-value pairs. There is a key `position` which sets `\l_@@_position_int` but it's faster to do the affectation right now.

```

6818     {
6819       \int_set:Nn \l_@@_position_int { ##1 }
6820       \l_@@_draw_hrulerule:
6821     }
6822   }
6823 }
6824 }

```

The command `\l_@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6825 \cs_set:Npn \l_@@_Hline: { \noalign \bgroup \l_@@_Hline_i:n { 1 } }

```

The argument of the command `\l_@@_Hline_i:n` is the number of successive `\Hline` found.

```

6826 \cs_set:Npn \l_@@_Hline_i:n #1
6827 {
6828   \peek_remove_spaces:n
6829   {
6830     \peek_meaning:NNTF \Hline
6831     { \l_@@_Hline_ii:nn { #1 + 1 } }
6832     { \l_@@_Hline_iii:n { #1 } }
6833   }
6834 }
6835 \cs_set:Npn \l_@@_Hline_ii:nn #1 #2 { \l_@@_Hline_i:n { #1 } }
6836 \cs_set:Npn \l_@@_Hline_iii:n #1
6837 { \l_@@_collect_options:n { \l_@@_Hline_iv:nn { #1 } } }
6838 \cs_set_protected:Npn \l_@@_Hline_iv:nn #1 #2
6839 {
6840   \l_@@_compute_rule_width:n { multiplicity = #1 , #2 }
6841   \skip_vertical:N \l_@@_rule_width_before_dim
6842   \tl_gput_right:Ne \g_@@_rules_tl
6843   {

```

With the keys of `nicematrix / rules-after` we would write:

```
\@@_draw_hrrule:n
{
  multiplicity = #1 ,
  position = \int_eval:n { \c@iRow + 1 } ,
  total-width = \dim_use:N \l_@@_rule_width_before_dim ,
  #2
}
```

We will use a version slightly more efficient:

```
6844 {
6845   \int_compare:nNt { #1 } > 1 { \@@_set_multiplicity:n { #1 } }
6846   \int_set:Nn \l_@@_position_int { \int_eval:n { \c@iRow + 1 } }
6847   \dim_set:Nn \l_@@_rule_width_after_dim
6848     { \dim_use:N \l_@@_rule_width_before_dim }
6849   \@@_draw_hrrule:n { #2 }
6850 }
6851 }
6852 \egroup
6853 }
```

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`).

```
6854 \cs_new_protected:Npn \@@_custom_line:n #1
6855 {
6856   \str_clear:N \l_@@_letter_str
6857   \str_clear:N \l_@@_command_str
6858   \str_clear:N \l_@@_ccommand_str
6859   \tl_clear_new:N \l_@@_other_keys_tl
6860   \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
6861   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
6862   {
6863     \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }
```

We delete the last character which is a space.

```
6864   \str_set:Ne \l_@@_command_str
6865     { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
6866   }
6867   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
6868   {
6869     \str_set:Ne \l_@@_ccommand_str
6870       { \str_tail:N \l_@@_ccommand_str }
6871     \str_set:Ne \l_@@_ccommand_str
6872       { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
6873   }
```

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.

```
6874 \bool_lazy_all:nTF
6875 {
6876   { \str_if_empty_p:N \l_@@_letter_str }
6877   { \str_if_empty_p:N \l_@@_command_str }
6878   { \str_if_empty_p:N \l_@@_ccommand_str }
6879 }
6880 { \@@_error:n { No~letter~and~no~command } }
```

```

6881     { \l_@@_custom_line_i:o \l_@@_other_keys_tl }
6882   }
6883 \keys_define:nn { nicematrix / custom-line }
6884 {
6885   letter .str_set:N = \l_@@_letter_str ,
6886   letter .value_required:n = true ,
6887   command .str_set:N = \l_@@_command_str ,
6888   command .value_required:n = true ,
6889   ccommand .str_set:N = \l_@@_ccommand_str ,
6890   ccommand .value_required:n = true
6891 }

6892 \cs_new_protected:Npn \l_@@_custom_line_i:n #1
6893 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6894   \bool_set_false:N \l_@@_tikz_rule_bool
6895   \bool_set_false:N \l_@@_dotted_rule_bool
6896   \bool_set_false:N \l_@@_color_bool
6897   \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6898   \bool_if:NT \l_@@_tikz_rule_bool
6899   {
6900     \IfPackageLoadedF { tikz }
6901     { \@@_error:n { tikz-in-custom-line-without-tikz } }
6902     \bool_if:NT \l_@@_color_bool
6903     { \@@_error:n { color-in-custom-line-with-tikz } }
6904   }
6905   \bool_if:NT \l_@@_dotted_rule_bool
6906   {
6907     \int_compare:nNnT \l_@@_multiplicity_int > 1
6908     { \@@_error:n { key-multiplicity-with-dotted } }
6909   }
6910   \str_if_empty:NF \l_@@_letter_str
6911   {
6912     \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6913     { \@@_error:n { Several~letters } }
6914     {
6915       \tl_if_in:NoTF
6916       \c_@@_forbidden_letters_str
6917       \l_@@_letter_str
6918       { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6919     }

```

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.

```

6920       \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6921       { \@@_v_custom_line:nn { #1 } }
6922     }
6923   }
6924 }
6925 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6926 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6927 }
6928 \cs_generate_variant:Nn \l_@@_custom_line_i:n { o }
6929 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6930 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\l_@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will

also be used further with other sets of keys (for instance `{nicematrix/rules-after}`). That's why the following set of keys has some keys which are no-op.

```

6931 \keys_define:nn { nicematrix / custom-line-bis }
6932 {
6933   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6934   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6935   color .value_required:n = true ,
6936   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6937   tikz .value_required:n = true ,
6938   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6939   dotted .value_forbidden:n = true ,
6940   total-width .code:n = { } ,
6941   total-width .value_required:n = true ,
6942   sep-color .code:n = { } ,
6943   sep-color .value_required:n = true ,
6944   unknown .code:n =
6945     \@@_unknown_key:nn
6946     { nicematrix / custom-line-bis }
6947     { Unknown-key-for~custom-line }
6948 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6949 \bool_new:N \l_@@_dotted_rule_bool
6950 \bool_new:N \l_@@_tikz_rule_bool
6951 \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).

```

6952 \keys_define:nn { nicematrix / custom-line-width }
6953 {
6954   multiplicity .int_set:N = \l_@@_tmpc_int ,
6955   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6956   total-width .code:n = \dim_set:Nn \l_@@_rule_width_before_dim { #1 }
6957                       \bool_set_true:N \l_@@_total_width_bool ,
6958   total-width .value_required:n = true ,
6959   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6960 }

```

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 `\@@_draw_hrule:n` (which is in the internal `\CodeAfter`).

```

6961 \cs_new_protected:Npn \@@_h_custom_line:n #1
6962 {

```

We use `\cs_set:cpn` and not `\cs_new:cpn` because we want a local definition. Moreover, the command must *not* be protected since it begins with `\noalign` (which is in `\Hline`).

```

6963   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6964   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6965 }

```

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 `\@@_draw_hrule:n` (which is in the internal `\CodeAfter`).

```

6966 \cs_new_protected:Npn \@@_c_custom_line:n #1
6967 {

```


Here, we need an expandable command since it begins with an `\noalign`.

```

6968 \exp_args:Nc \DeclareExpandableDocumentCommand
6969 { nicematrix - \l_@@_ccommand_str }
6970 { 0 { } m }
6971 {
6972   \noalign
6973   {
6974     \@@_compute_rule_width:n { #1 , ##1 }
6975     \skip_vertical:n \l_@@_rule_width_before_dim
6976     \clist_map_inline:nn
6977       { ##2 }
6978       { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6979   }
6980 }
6981 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6982 }

```

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*.

```

6983 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6984 {
6985   \tl_if_in:nnTF { #2 } { - }
6986   { \@@_cut_on_hyphen:w #2 \q_stop }
6987   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6988   \tl_gput_right:Ne \g_@@_rules_tl
6989   {
6990     \@@_draw_hrule:n
6991     {
6992       #1 ,
6993       start = \l_tmpa_tl ,
6994       end = \l_tmpb_tl ,
6995       position = \int_eval:n { \c@iRow + 1 } ,
6996       total-width = \dim_use:N \l_@@_rule_width_before_dim
6997     }
6998   }
6999 }

7000 \cs_new_protected:Npn \@@_compute_rule_width:n #1
7001 {
7002   \bool_set_false:N \l_@@_tikz_rule_bool
7003   \bool_set_false:N \l_@@_total_width_bool
7004   \bool_set_false:N \l_@@_dotted_rule_bool
7005   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
7006   \bool_if:NF \l_@@_total_width_bool
7007   {
7008     \bool_if:NTF \l_@@_dotted_rule_bool
7009     { \dim_set:Nn \l_@@_rule_width_before_dim { 2 \l_@@_xdots_radius_dim } }
7010     {
7011       \bool_if:NF \l_@@_tikz_rule_bool
7012       {
7013         \dim_set:Nn \l_@@_rule_width_before_dim
7014         {
7015           \arrayrulewidth * \l_@@_tmpc_int
7016           + \doublerulesep * ( \l_@@_tmpc_int - 1 )
7017         }
7018       }
7019     }
7020   }
7021 }

```

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

```

7022 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
7023 {

```

```

7024 \str_if_eq:nnTF { #2 } { [ ]
7025   { \@@_v_custom_line_i:nw { #1 } [ ]
7026   { \@@_v_custom_line_ii:nn { #2 } { #1 } }
7027 }
7028 \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
7029 { \@@_v_custom_line:nn { #1 , #2 } }
7030 \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
7031 {
7032   \@@_compute_rule_width:n { #2 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

7033 \tl_gput_right:Ne \g_@@_array_preamble_tl
7034 {
7035   \exp_not:N !
7036   { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_before_dim } }
7037 }
7038 \tl_gput_right:Ne \g_@@_rules_tl
7039 {

```

Here is a version with the keys of `rules-after`.

```

\@@_draw_vrule:n
{
  #2 ,
  position = \int_eval:n { \c@jCol + 1 } ,
  total-width = \dim_use:N \l_@@_rule_width_before_dim
}

```

However, you will use a slightly more efficient version.

```

7040 {
7041   \dim_set:Nn \l_@@_rule_width_after_dim
7042   { \dim_use:N \l_@@_rule_width_before_dim }
7043   \int_set:Nn \l_@@_position_int { \int_eval:n { \c@jCol + 1 } }
7044   \@@_draw_vrule:n { #2 }
7045 }
7046 }
7047 \@@_rec_preamble:n #1
7048 }

7049 \@@_custom_line:n
7050 { letter = : , command = \hdottedline , ccommand = \cdottedline, dotted }

7051 \AtBeginDocument
7052 {
7053   \IfPackageLoadedT { tikz }
7054   {
7055     \cs_if_exist:cTF { tikz@library@decorations@loaded }
7056     {
7057       \@@_custom_line:n
7058       {
7059         letter = ; ,
7060         command = \hdashedline ,
7061         ccommand = \cdashedline ,
7062         tikz =
7063         {
7064           dash-pattern = on~4~pt~off~2pt,
7065           dash-expand-off ,
7066           line-cap = butt
7067         } ,
7068         total-width = \pgflinewidth
7069       }
7070     }
7071     {
7072       \@@_custom_line:n
7073       {

```

```

7074         letter = ; ,
7075         command = \hdashedline ,
7076         ccommand = \cdashedline ,
7077         tikz = { dash-pattern = on~4~pt~off~2pt , line-cap = butt} ,
7078         total-width = \pgflinewidth
7079     }
7080 }
7081 }
7082 }

```

The key default-line

```

7083 \keys_define:nn { nicematrix / default-line }
7084 {
7085     multiplicity .int_set:N = \l_@@_multiplicity_int ,
7086     color .code:n = \bool_set_true:N \l_@@_color_bool ,
7087     color .value_required:n = true ,
7088     tikz .code:n =
7089         \bool_set_true:N \l_@@_tikz_rule_bool
7090         \tl_set:Nn \l_@@_tikz_rule_tl { #1 } ,
7091     tikz .value_required:n = true ,
7092     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7093     dotted .value_forbidden:n = true ,
7094     total-width .code:n = { } ,
7095     total-width .value_required:n = true ,
7096     sep-color .code:n = { } ,
7097     sep-color .value_required:n = true ,
7098     unknown .code:n =
7099         \@@_unknown_key:nn
7100             { nicematrix / default-line }
7101             { Unknown-key-for~default-line }
7102 }
7103 \cs_new_protected:Npn \@@_default_line:n #1
7104 {
7105     \bool_set_false:N \l_@@_tikz_rule_bool
7106     \bool_set_false:N \l_@@_dotted_rule_bool
7107     \bool_set_false:N \l_@@_color_bool
7108     \keys_set:nn { nicematrix / default-line } { #1 }
7109     \bool_if:NT \l_@@_tikz_rule_bool
7110     {
7111         \IfPackageLoadedF { tikz }
7112         { \@@_error:n { tikz~in~custom~line~without~tikz } }
7113         \bool_if:NT \l_@@_color_bool
7114         { \@@_error:n { color~in~custom~line~with~tikz } }
7115     }
7116     \bool_if:NT \l_@@_dotted_rule_bool
7117     {
7118         \int_compare:nNnT \l_@@_multiplicity_int > 1
7119         { \@@_error:n { key~multiplicity~with~dotted } }
7120     }
7121     \bool_if:NTF \l_@@_tikz_rule_bool
7122     {
7123         \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
7124         \socket_assign_plug:nn { nicematrix / hsegment } { tikz }
7125     }
7126     {
7127         \bool_if:NTF \l_@@_dotted_rule_bool
7128         {
7129             \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
7130             \socket_assign_plug:nn { nicematrix / hsegment } { dotted }
7131         }
7132         {
7133             \bool_if:NTF \l_@@_multiple_rule_bool

```

```

7134         {
7135             \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
7136             \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
7137         }
7138     }
7139 }
7140 }

```

The key hvlines

The following command tests whether the current position in the array (given by `\l_tmpa_tl` for the row and `\l_tmpb_tl` for the column) would provide an horizontal rule towards the right in the block delimited by the four arguments #1, #2, #3 and #4. If this rule would be in the block (it must not be drawn), the boolean `\g_tmpa_bool` is set to false.

```

7141 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
7142 {
7143     \int_compare:nNtT \l_tmpa_tl > { #1 }
7144     {
7145         \int_compare:nNtT \l_tmpa_tl < { #3 + 1 }
7146         {
7147             \int_compare:nNtT \l_tmpb_tl > { #2 - 1 }
7148             {
7149                 \int_compare:nNtT \l_tmpb_tl < { #4 + 1 }
7150                 { \bool_gset_false:N \g_tmpa_bool }
7151             }
7152         }
7153     }
7154 }

```

The same for vertical rules.

```

7155 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
7156 {
7157     \int_compare:nNtT \l_tmpa_tl > { #1 - 1 }
7158     {
7159         \int_compare:nNtT \l_tmpa_tl < { #3 + 1 }
7160         {
7161             \int_compare:nNtT \l_tmpb_tl > { #2 }
7162             {
7163                 \int_compare:nNtT \l_tmpb_tl < { #4 + 1 }
7164                 { \bool_gset_false:N \g_tmpa_bool }
7165             }
7166         }
7167     }
7168 }
7169 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
7170 {
7171     \int_compare:nNtT \l_tmpb_tl > { #2 - 1 }
7172     {
7173         \int_compare:nNtT \l_tmpb_tl < { #4 + 1 }
7174         {
7175             \int_compare:nNtTF \l_tmpa_tl = { #1 }
7176             { \bool_gset_false:N \g_tmpa_bool }
7177             {
7178                 \int_compare:nNtT \l_tmpa_tl = { #3 + 1 }
7179                 { \bool_gset_false:N \g_tmpa_bool }
7180             }
7181         }
7182     }
7183 }
7184 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
7185 {
7186     \int_compare:nNtT \l_tmpa_tl > { #1 - 1 }
7187     {

```

```

7188     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
7189     {
7190         \int_compare:nNnTF \l_tmpb_tl = { #2 }
7191         { \bool_gset_false:N \g_tmpa_bool }
7192         {
7193             \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
7194             { \bool_gset_false:N \g_tmpa_bool }
7195         }
7196     }
7197 }
7198 }

```

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.

```

7199 \cs_new_protected:Npn \@@_compute_corners:
7200 {
7201     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7202     { \@@_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`).

```

7203     \clist_clear:N \l_@@_corners_cells_clist
7204     \clist_map_inline:Nn \l_@@_corners_cells_clist
7205     {
7206         \str_case:nnF { ##1 }
7207         {
7208             { NW }
7209             { \@@_compute_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
7210             { NE }
7211             { \@@_compute_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
7212             { SW }
7213             { \@@_compute_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
7214             { SE }
7215             { \@@_compute_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
7216         }
7217         { \@@_error:nn { bad~corner } { ##1 } }
7218     }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

7219     \clist_if_empty:NF \l_@@_corners_cells_clist
7220     {

```

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.

```

7221     \tl_gput_right:Ne \g_@@_aux_tl
7222     {
7223         \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
7224         { \l_@@_corners_cells_clist }
7225     }
7226 }
7227 }

```

```

7228 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
7229 {
7230   \int_step_inline:nnn { #1 } { #3 }
7231   {
7232     \int_step_inline:nnn { #2 } { #4 }
7233     { \cs_set_nopar:cpn { @@ _ block _ ##1 - ###1 } { } }
7234   }
7235 }

7236 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
7237 {
7238   \cs_if_exist:cTF
7239   { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
7240   \prg_return_true:
7241   \prg_return_false:
7242 }

```

“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_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.

```

7243 \cs_new_protected:Npn \@@_compute_corner:nnnnnn #1 #2 #3 #4 #5 #6
7244 {

```

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.

```

7245   \bool_set_false:N \l_tmpa_bool
7246   \int_zero_new:N \l_@@_last_empty_row_int
7247   \int_set:Nn \l_@@_last_empty_row_int { #1 }
7248   \int_step_inline:nnnn { #1 } { #3 } { #5 }
7249   {
7250     \bool_lazy_or:nnTF
7251     {
7252       \cs_if_exist_p:c
7253       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
7254     }
7255     { \@@_if_in_block_p:nn { ##1 } { #2 } }
7256     { \bool_set_true:N \l_tmpa_bool }
7257   }
7258   \bool_if:NF \l_tmpa_bool
7259   { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
7260 }
7261 }

```

Now, you determine the last empty cell in the row of number 1.

```

7262   \bool_set_false:N \l_tmpa_bool
7263   \int_zero_new:N \l_@@_last_empty_column_int
7264   \int_set:Nn \l_@@_last_empty_column_int { #2 }
7265   \int_step_inline:nnnn { #2 } { #4 } { #6 }
7266   {
7267     \bool_lazy_or:nnTF
7268     {

```

```

7269     \cs_if_exist_p:c
7270     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
7271   }
7272   { \@@_if_in_block_p:nn { #1 } { ##1 } }
7273   { \bool_set_true:N \l_tmpa_bool }
7274   {
7275     \bool_if:NF \l_tmpa_bool
7276     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
7277   }
7278 }

```

Now, we loop over the rows.

```

7279   \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
7280   {

```

We treat the row number ##1 with another loop.

```

7281     \bool_set_false:N \l_tmpa_bool
7282     \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
7283     {
7284       \bool_lazy_or:nnTF
7285       { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7286       { \@@_if_in_block_p:nn { ##1 } { #####1 } }
7287       { \bool_set_true:N \l_tmpa_bool }
7288     {
7289       \bool_if:NF \l_tmpa_bool
7290       {
7291         \int_set:Nn \l_@@_last_empty_column_int { #####1 }
7292         \clist_put_right:Nn
7293         \l_@@_corners_cells_clist
7294         { ##1 - #####1 }
7295         \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7296       }
7297     }
7298   }
7299 }
7300 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

7301 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
7302 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }

```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient:
`\clist_if_in:NcT \l_@@_corners_cells_clist { #1 } ...`

24 The environment {NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```

7303 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

7304 \keys_define:nn { nicematrix / NiceMatrixBlock }
7305 {
7306   auto-columns-width .code:n =
7307   {
7308     \bool_set_true:N \l_@@_block_auto_columns_width_bool
7309     \dim_gzero_new:N \g_@@_max_cell_width_dim
7310     \bool_set_true:N \l_@@_auto_columns_width_bool
7311   }
7312 }

```

```

7313 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
7314 {
7315   \int_gincr:N \g_@@_NiceMatrixBlock_int
7316   \dim_zero:N \l_@@_columns_width_dim
7317   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
7318   \bool_if:NT \l_@@_block_auto_columns_width_bool
7319   {
7320     \cs_if_exist:cT
7321     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
7322     {
7323       \dim_set:Nn \l_@@_columns_width_dim
7324       {
7325         \use:c
7326         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7327       }
7328     }
7329   }
7330 }

```

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`).

```

7331 {
7332   \legacy_if:nTF { measuring@ }

```

If `{NiceMatrixBlock}` is used in an environment of `amsmath` such as `{align}`: cf. question 694957 on TeX StackExchange. The most important line in that case is the following one.

```

7333   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
7334   {
7335     \bool_if:NT \l_@@_block_auto_columns_width_bool
7336     {
7337       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
7338       \iow_shipout:Ne \@mainaux
7339       {
7340         \cs_gset:cpn
7341         { @@ _ 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.

```

7342       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7343     }
7344     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
7345   }
7346 }
7347 \ignorespacesafterend
7348 }

```

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).

```

7349 \cs_new_protected:Npn \@@_create_extra_nodes:
7350 {
7351   \bool_if:nTF \l_@@_medium_nodes_bool
7352   {
7353     \bool_if:NTF \l_@@_no_cell_nodes_bool
7354     { \@@_error:n { extra-nodes~with~no~cell~nodes } }
7355     {
7356       \bool_if:NTF \l_@@_large_nodes_bool

```



```

7357         \@@_create_medium_and_large_nodes:
7358         \@@_create_medium_nodes:
7359     }
7360 }
7361 {
7362     \bool_if:NT \l_@@_large_nodes_bool
7363     {
7364         \bool_if:NTF \l_@@_no_cell_nodes_bool
7365         { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7366         \@@_create_large_nodes:
7367     }
7368 }
7369 }

```

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`.

```

7370 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7371 {
7372     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7373     {
7374         \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7375         \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7376         \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7377         \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7378     }
7379     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7380     {
7381         \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7382         \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7383         \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7384         \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7385     }

```

We begin the two nested loops over the rows and the columns of the array.

```

7386     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7387     {
7388         \int_step_variable:nnNn
7389         \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.

```

7390     {
7391         \cs_if_exist:cT
7392         { 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`.

```

7393     {
7394         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7395         \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7396         { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7397         \seq_if_in:Nef \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7398         {
7399             \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7400             { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7401         }

```

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`.

```

7402         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7403         \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7404         { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
7405         \seq_if_in:Nef \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7406         {
7407             \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7408             { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
7409         }
7410     }
7411 }
7412 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7413 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7414 {
7415     \dim_compare:nNnT
7416     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7417     {
7418         \@@_qpoint:n { row - \@@_i: - base }
7419         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7420         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7421     }
7422 }
7423 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7424 {
7425     \dim_compare:nNnT
7426     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7427     {
7428         \@@_qpoint:n { col - \@@_j: }
7429         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7430         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7431     }
7432 }
7433 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7434 \cs_new_protected:Npn \@@_create_medium_nodes:
7435 {
7436     \pgfpicture
7437     \pgfrememberpicturepositiononpagetrue
7438     \pgf@relevantforpicturesizefalse
7439     \@@_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”.

```

7440     \tl_set:Nn \l_@@_suffix_tl { -medium }
7441     \@@_create_nodes:

```

```

7442 \endpgfpicture
7443 }

```

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:`.

```

7444 \cs_new_protected:Npn \@@_create_large_nodes:
7445 {
7446   \pgfpicture
7447   \pgfrememberpicturepositiononpagetrue
7448   \pgf@relevantforpicturesizefalse
7449   \@@_computations_for_medium_nodes:
7450   \@@_computations_for_large_nodes:
7451   \tl_set:Nn \l_@@_suffix_tl { - large }
7452   \@@_create_nodes:
7453   \endpgfpicture
7454 }
7455 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7456 {
7457   \pgfpicture
7458   \pgfrememberpicturepositiononpagetrue
7459   \pgf@relevantforpicturesizefalse
7460   \@@_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”.

```

7461   \tl_set:Nn \l_@@_suffix_tl { - medium }
7462   \@@_create_nodes:
7463   \@@_computations_for_large_nodes:
7464   \tl_set:Nn \l_@@_suffix_tl { - large }
7465   \@@_create_nodes:
7466   \endpgfpicture
7467 }

```

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.

```

7468 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7469 {
7470   \int_set:Nn \l_@@_first_row_int 1
7471   \int_set:Nn \l_@@_first_col_int 1

```

We have to change the values of all the dimensions `l_@@_row_i_min_dim`, `l_@@_row_i_max_dim`, `l_@@_column_j_min_dim` and `l_@@_column_j_max_dim`.

```

7472   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7473   {
7474     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7475     {
7476       (
7477         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7478         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7479       )
7480       / 2
7481     }
7482     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7483     { l_@@_row _ \@@_i: _ min _ dim }
7484   }
7485   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:

```

¹⁵If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7486 {
7487   \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7488   {
7489     (
7490       \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7491       \dim_use:c
7492         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7493     )
7494     / 2
7495   }
7496   \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7497   { l_@@_column _ \@@_j: _ max _ dim }
7498 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7499 \dim_sub:cn
7500   { l_@@_column _ 1 _ min _ dim }
7501   \l_@@_left_margin_dim
7502 \dim_add:cn
7503   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7504   \l_@@_right_margin_dim
7505 }

```

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).

```

7506 \cs_new_protected:Npn \@@_create_nodes:
7507 {
7508   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7509   {
7510     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7511     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7512       \@@_pgf_rect_node:nnnnn
7513       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7514       { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7515       { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7516       { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7517       { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7518       \str_if_empty:NF \l_@@_name_str
7519       {
7520         \pgfnodealias
7521         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7522         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7523       }
7524     }
7525   }
7526 \int_step_inline:nn \c@iRow
7527 {
7528   \pgfnodealias
7529   { \@@_env: - ##1 - last \l_@@_suffix_tl }
7530   { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7531 }
7532 \int_step_inline:nn \c@jCol
7533 {
7534   \pgfnodealias
7535   { \@@_env: - last - ##1 \l_@@_suffix_tl }
7536   { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7537 }

```

```

7538 \pgfnodealias
7539 { \@@_env: - last - last \l_@@_suffix_tl }
7540 { \@@_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 .

```

7541 \seq_map_pairwise_function:NNN
7542 \g_@@_multicolumn_cells_seq
7543 \g_@@_multicolumn_sizes_seq
7544 \@@_node_for_multicolumn:nn
7545 }

7546 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7547 {
7548   \cs_set_nopar:Npn \@@_i: { #1 }
7549   \cs_set_nopar:Npn \@@_j: { #2 }
7550 }

```

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”).

```

7551 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7552 {
7553   \@@_extract_coords_values: #1 \q_stop
7554   \@@_pgf_rect_node:nnnnn
7555   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7556   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7557   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7558   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: + #2 - 1 } _ max _ dim } }
7559   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7560   \str_if_empty:NF \l_@@_name_str
7561   {
7562     \pgfnodealias
7563     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7564     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7565   }
7566 }

```

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).

```

7567 \keys_define:nn { nicematrix / Block / FirstPass }
7568 {
7569   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7570           \bool_set_true:N \l_@@_p_block_bool ,
7571   j .value_forbidden:n = true ,
7572   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7573   l .value_forbidden:n = true ,
7574   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7575   r .value_forbidden:n = true ,
7576   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7577   c .value_forbidden:n = true ,

```

```

7578 L.code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7579 L.value_forbidden:n = true ,
7580 R.code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7581 R.value_forbidden:n = true ,
7582 C.code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7583 C.value_forbidden:n = true ,
7584 t.code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7585 t.value_forbidden:n = true ,
7586 T.code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7587 T.value_forbidden:n = true ,
7588 b.code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7589 b.value_forbidden:n = true ,
7590 B.code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7591 B.value_forbidden:n = true ,
7592 m.code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7593 m.value_forbidden:n = true ,
7594 v-center.meta:n = m ,
7595 p.code:n = \bool_set_true:N \l_@@_p_block_bool ,
7596 p.value_forbidden:n = true ,
7597 respect-arraystretch.code:n =
7598   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7599 respect-arraystretch.value_forbidden:n = true ,
7600 color.code:n =
7601   \@@_color:n { #1 }
7602   \tl_set_rescan:Nnn
7603     \l_@@_draw_tl
7604     { \char_set_catcode_other:N ! }
7605     { #1 } ,
7606   color.value_required:n = true ,
7607 }

```

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.

```

7608 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7609 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7610 {

```

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).

```

7611   \tl_if_blank:nTF { #2 }
7612     { \@@_Block_ii:nnnnn 1 1 }
7613     {
7614       \tl_if_in:nnTF { #2 } { - }
7615       {
7616         \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7617         \@@_Block_i_czech:w \@@_Block_i:w
7618         #2 \q_stop
7619       }
7620       {
7621         \@@_error:nn { Bad~argument~for~Block } { #2 }
7622         \@@_Block_ii:nnnnn 1 1
7623       }
7624     }
7625   { #1 } { #3 } { #4 }
7626   \ignorespaces
7627 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7628 \cs_new:Npn \@@_Block_i:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }

```

With `babel` with the key `czech`, the character `-` (hyphen) is active. That's why we need a special version. Remark that we could not use a preprocessor in the command `\@@_Block:` to do the job because the command `\@@_Block:` is defined with the command `\NewExpandableDocumentCommand`.

```

7629 {
7630   \char_set_catcode_active:N -
7631   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7632 }

```

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.

```

7633 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7634 {

```

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).

```

7635   \bool_lazy_or:nnTF
7636     { \tl_if_blank_p:n { #1 } }
7637     { \str_if_eq_p:ee { * } { #1 } }
7638     { \int_set:Nn \l_tmpa_int { 100 } }
7639     { \int_set:Nn \l_tmpa_int { #1 } }
7640   \bool_lazy_or:nnTF
7641     { \tl_if_blank_p:n { #2 } }
7642     { \str_if_eq_p:ee { * } { #2 } }
7643     { \int_set:Nn \l_tmpb_int { 100 } }
7644     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7645   \int_compare:nNnTF \l_tmpb_int = 1
7646   {
7647     \tl_if_empty:NTF \l_@@_hpos_cell_tl
7648       { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7649       { \str_set:Nn \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7650   }
7651   { \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.

```

7652   \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7653   \tl_set:Ne \l_tmpa_tl
7654   {
7655     { \int_use:N \c@iRow }
7656     { \int_use:N \c@jCol }
7657     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7658     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7659   }

```

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).

```

7660   \bool_set_false:N \l_tmpa_bool
7661   \bool_if:NT \l_@@_amp_in_blocks_bool

```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

7662     { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7663     \bool_case:nF
7664     {
7665         \l_tmpa_bool                { \@@_Block_vii:eenenn }
7666         \l_@@_p_block_bool          { \@@_Block_vi:eenenn }

```

For the blocks mono-column, we will compose right away in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a X column, we should not do that since the width is determined by another way. This should be the same for the p, m and b columns and we should modify that point. However, for the X column, it's imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

7667         \l_@@_X_bool                { \@@_Block_v:eenenn }
7668         { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eenenn }
7669         { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eenenn }
7670         { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eenenn }
7671     }
7672     { \@@_Block_v:eenenn }
7673     { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7674 }

```

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.

```

7675 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7676 {
7677     \int_gincr:N \g_@@_block_box_int
7678     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
7679     \cs_set_eq:NN \rowcolor \@@_rowcolor_error
7680     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7681     {
7682         \tl_gput_right:Ne \g_@@_rules_tl
7683         {
7684             \@@_draw_diagbox:nnnnnn
7685             { \int_use:N \c@iRow }
7686             { \int_use:N \c@jCol }
7687             { \int_eval:n { \c@iRow + #1 - 1 } }
7688             { \int_eval:n { \c@jCol + #2 - 1 } }
7689             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7690             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7691         }
7692     }
7693     \box_gclear_new:c
7694     { g_@@_block_box_int _ \int_use:N \g_@@_block_box_int _ box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7695     \hbox_gset:cn
7696     { g_@@_block_box_int _ \int_use:N \g_@@_block_box_int _ box }
7697     {

```

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`).

```

7698      \tl_if_empty:NTF \l_@@_color_tl
7699      { \int_compare:nNnT { #2 } = 1 \set@color }
7700      { \@@_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`.

```

7701      \int_compare:nNnT { #1 } = 1
7702      {
7703          \int_if_zero:nTF \c@iRow
7704          {

```

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}$

```

```

7705      \cs_set_eq:NN \Block \@@_NullBlock:
7706      \l_@@_code_for_first_row_tl
7707      }
7708      {
7709          \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7710          {
7711              \cs_set_eq:NN \Block \@@_NullBlock:
7712              \l_@@_code_for_last_row_tl
7713          }
7714      }
7715      \g_@@_row_style_tl
7716      }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7717      \@@_reset_arraystretch:
7718      \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7719      #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.).

```

7720      \@@_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}`.

```

7721 \bool_if:NTF \l_@@_tabular_bool
7722 {
7723   \bool_lazy_all:nTF
7724   {
7725     { \int_compare_p:nNn { #2 } = 1 }

```

Remind that, when the column has not a fixed width, the dimension `\l_@@_col_width_dim` has the conventional value of -1 cm.

```

7726     { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7727     { ! \g_@@_rotate_bool }
7728   }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7729   {
7730     \use:e
7731     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7732     \exp_not:N \begin { minipage }
7733     [ \str_lowercase:f \l_@@_vpos_block_str ]
7734     { \l_@@_col_width_dim }
7735     \str_case:on \l_@@_hpos_block_str
7736     { c \centering r \raggedleft l \raggedright }
7737   }
7738   #5
7739   \end { minipage }
7740 }

```

In the other cases, we use a `{tabular}`.

```

7741   {
7742     \use:e
7743     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7744     \exp_not:N \begin { tabular }
7745     [ \str_lowercase:f \l_@@_vpos_block_str ]
7746     { @ { } \l_@@_hpos_block_str @ { } }
7747   }
7748   #5
7749   \end { tabular }
7750 }
7751 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is `false`). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7752   {
7753     $ % $
7754     \use:e
7755     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7756     \exp_not:N \begin { array }
7757     [ \str_lowercase:f \l_@@_vpos_block_str ]
7758     { @ { } \l_@@_hpos_block_str @ { } }
7759   }
7760   #5
7761   \end { array }
7762   $ % $
7763 }
7764 }

```

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).

```

7765 \bool_if:NT \g_@@_rotate_bool \@@_rotate_box_of_block:

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

7766     \int_compare:nNnT { #2 } = 1
7767     {
7768         \dim_gset:Nn \g_@@_blocks_wd_dim
7769         {
7770             \dim_max:nn
7771             \g_@@_blocks_wd_dim
7772             {
7773                 \box_wd:c
7774                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7775             }
7776         }
7777     }

```

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.

```

7778     \int_compare:nNnT { #1 } = 1
7779     {
7780         \bool_lazy_any:nT
7781         {
7782             { \str_if_empty_p:N \l_@@_vpos_block_str }
7783             { \str_if_eq_p:ee t \l_@@_vpos_block_str }
7784             { \str_if_eq_p:ee b \l_@@_vpos_block_str }
7785         }
7786         \@@_adjust_blocks_ht_dp:
7787     }
7788     \seq_gput_right:Ne \g_@@_blocks_seq
7789     {
7790         \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.

```

7791     {
7792         \exp_not:n { #3 } ,
7793         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7794         \bool_if:NT \g_@@_rotate_bool
7795         {
7796             \bool_if:NTF \g_@@_rotate_c_bool
7797             { m }
7798             {
7799                 \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7800                 { T }
7801             }
7802         }
7803     }
7804     {
7805         \box_use_drop:c
7806         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7807     }
7808 }
7809 \bool_set_false:N \g_@@_rotate_c_bool
7810 }

7811 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7812 {
7813     \dim_gset:Nn \g_@@_blocks_ht_dim
7814     {

```

```

7815     \dim_max:nn
7816     \g_@@_blocks_ht_dim
7817     {
7818         \box_ht:c
7819         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7820     }
7821 }
7822 \dim_gset:Nn \g_@@_blocks_dp_dim
7823 {
7824     \dim_max:nn
7825     \g_@@_blocks_dp_dim
7826     {
7827         \box_dp:c
7828         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7829     }
7830 }
7831 }

7832 \cs_new:Npn \@@_adjust_hpos_rotate:
7833 {
7834     \bool_if:NT \g_@@_rotate_bool
7835     {
7836         \str_set:Ne \l_@@_hpos_block_str
7837         {
7838             \bool_if:NTF \g_@@_rotate_c_bool
7839             c
7840             {
7841                 \str_case:onF \l_@@_vpos_block_str
7842                 { b l B l t r T r }
7843                 {
7844                     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
7845                     r
7846                     l
7847                 }
7848             }
7849         }
7850     }
7851 }
7852 \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*.

```

7853 \cs_new_protected:Npn \@@_rotate_box_of_block:
7854 {
7855     \box_grotate:cn
7856     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7857     { 90 }
7858     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7859     {
7860         \vbox_gset_top:cn
7861         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7862         {
7863             \skip_vertical:n { 0.8 ex }
7864             \box_use:c
7865             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7866         }
7867     }
7868     \bool_if:NT \g_@@_rotate_c_bool
7869     {
7870         \hbox_gset:cn
7871         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7872         {

```

```

7873         $ % $
7874         \vcenter
7875         {
7876             \box_use:c
7877             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7878         }
7879         $ % $
7880     }
7881 }
7882 }

```

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.

```

7883 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5
7884 {
7885     \seq_gput_right:Ne \g_@@_blocks_seq
7886     {
7887         \l_tmpa_tl
7888         { \exp_not:n { #3 } }
7889         {
7890             \bool_if:NTF \l_@@_tabular_bool
7891             {
7892

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7893     \@@_reset_arraystretch:
7894     \exp_not:n
7895     {
7896         \dim_zero:N \extrarowheight
7897         #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).

```

7898         \tag_if_active:T { \tag_stop:n { table } }
7899         \use:e
7900         {
7901             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7902             { @ { } \l_@@_hpos_block_str @ { } }
7903         }
7904         #5
7905         \end { tabular }
7906     }
7907 }
7908 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7909     {
7910     {

```

The following will be no-op when `respect-arraystretch` is in force.

```

7911     \@@_reset_arraystretch:
7912     \exp_not:n
7913     {
7914         \dim_zero:N \extrarowheight
7915         #4
7916         $ % $

```

```

7917         \use:e
7918         {
7919             \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7920             { @ { } \l_@@_hpos_block_str @ { } }
7921         }
7922         #5
7923         \end { array }
7924         $ % $
7925     }
7926 }
7927 }
7928 }
7929 }
7930 }
7931 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7932 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7933 {
7934     \seq_gput_right:Ne \g_@@_blocks_seq
7935     {
7936         \l_tmpa_tl
7937         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7938         { { \exp_not:n { #4 #5 } } }
7939     }
7940 }
7941 \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`.

```

7942 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7943 {
7944     \seq_gput_right:Ne \g_@@_blocks_seq
7945     {
7946         \l_tmpa_tl
7947         { \exp_not:n { #3 } }
7948         { \exp_not:n { #4 #5 } }
7949     }
7950 }
7951 \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).

```

7952 \keys_define:nn { nicematrix / Block / SecondPass }
7953 {
7954     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7955     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7956     tikz .code:n =
7957         \IfPackageLoadedTF { tikz }
7958         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7959         { \@@_error:n { tikz~key~without~tikz } } ,
7960     tikz .value_required:n = true ,
7961     fill .code:n =
7962         \tl_set_rescan:Nnn
7963         \l_@@_fill_tl
7964         { \char_set_catcode_other:N ! }
7965         { #1 } ,
7966     fill .value_required:n = true ,

```

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

```

7967 opacity .tl_set:N = \l_@@_opacity_tl ,
7968 opacity .value_required:n = true ,
7969 draw .code:n =
7970   \tl_set_rescan:Nnn
7971   \l_@@_draw_tl
7972   { \char_set_catcode_other:N ! }
7973   { #1 } ,
7974 draw .default:n = default ,
7975 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7976 rounded-corners .default:n = 4 pt ,
7977 color .code:n =
7978   \@@_color:n { #1 }
7979   \tl_set_rescan:Nnn
7980   \l_@@_draw_tl
7981   { \char_set_catcode_other:N ! }
7982   { #1 } ,
7983 borders .clist_set:N = \l_@@_borders_clist ,
7984 borders .value_required:n = true ,
7985 hvlines .meta:n = { vlines , hlines } ,
7986 vlines .bool_set:N = \l_@@_vlines_block_bool ,
7987 hlines .bool_set:N = \l_@@_hlines_block_bool ,
7988 rules/width .dim_set:N = \arrayrulewidth ,

```

The key `line-width` is now deprecated (replaced by `rules/width`).

```

7989 line-width .dim_set:N = \arrayrulewidth ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7990 j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7991   \bool_set_true:N \l_@@_p_block_bool ,
7992 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7993 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7994 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7995 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7996   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7997 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7998   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7999 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
8000   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8001 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
8002 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
8003 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
8004 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
8005 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
8006 m .value_forbidden:n = true ,
8007 v-center .meta:n = m ,
8008 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
8009 p .value_forbidden:n = true ,
8010 name .str_set:N = \l_@@_block_name_str ,
8011 name .value_required:n = true ,
8012 respect-arraystretch .code:n =
8013   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
8014 respect-arraystretch .value_forbidden:n = true ,
8015 transparent .bool_set:N = \l_@@_transparent_bool ,
8016 unknown .code:n =
8017   \@@_unknown_key:nn
8018   { nicematrix / Block / SecondPass }
8019   { Unknown-key-for-Block }
8020 }

```

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.

```

8021 \cs_new_protected:Npn \@@_draw_blocks:
8022 {
8023   \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
8024   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
8025 }
8026 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
8027 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

8028   \int_zero:N \l_@@_last_row_int
8029   \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”).

```

8030   \int_compare:nNnTF { #3 } > { 98 }
8031   { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
8032   { \int_set:Nn \l_@@_last_row_int { #3 } }
8033   \int_compare:nNnTF { #4 } > { 98 }
8034   { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
8035   { \int_set:Nn \l_@@_last_col_int { #4 } }
8036   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
8037   {
8038     \bool_lazy_and:nnTF
8039     \l_@@_preamble_bool
8040     {
8041       \int_compare_p:n
8042       { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
8043     }
8044     {
8045       \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
8046       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
8047       \@@_msg_redirect_name:nn { columns-not-used } { none }
8048     }
8049     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8050   }
8051   {
8052     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
8053     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8054     {

```

We expand the four first arguments of `\@@_Block_v:nnnnnn`.

```

8055       \use:e
8056       {
8057         \@@_Block_v:nnnnnn
8058         { #1 }
8059         { #2 }
8060         { \int_use:N \l_@@_last_row_int }
8061         { \int_use:N \l_@@_last_col_int }
8062       }
8063       { #5 }
8064       { #6 }
8065     }
8066   }
8067 }

```

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.


```

8068 \cs_new_protected:Npn \l_@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
8069 {

```

The group is for the keys.

```

8070 \group_begin:
8071 \int_compare:nNt { #1 } = { #3 }
8072 { \str_set:Nn \l_@@_vpos_block_str { t } }
8073 \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.

```

8074 \bool_if:NT \l_@@_amp_in_blocks_bool
8075 { \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool } }

8076 \bool_lazy_and:nnT
8077 \l_@@_vlines_block_bool
8078 { ! \l_@@_ampersand_bool }
8079 {
8080 \tl_gput_right:Ne \g_@@_rules_tl
8081 {
8082 \l_@@_vlines_block:nnnnn
8083 { \dim_use:N \arrayrulewidth }
8084 { #1 } { #2 } { #3 } { #4 }
8085 }
8086 }
8087 \bool_if:NT \l_@@_hlines_block_bool
8088 {
8089 \tl_gput_right:Ne \g_@@_rules_tl
8090 {
8091 \l_@@_hlines_block:nnnnn
8092 { \dim_use:N \arrayrulewidth }
8093 { #1 } { #2 } { #3 } { #4 }
8094 }
8095 }

```

The sequence of the positions of the blocks (excepted the blocks with the key hvlines) will be used when drawing the rules (in fact, there is also the \multicolumn and the \diagbox in that sequence).

```

8096 \bool_if:NF \l_@@_transparent_bool
8097 {
8098 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8099 { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
8100 }

8101 \tl_if_empty:NF \l_@@_draw_tl
8102 {
8103 \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
8104 { \@@_error:n { hlines-with-color } }
8105 \tl_gput_right:Nn \g_@@_rules_tl
8106 {
8107 \@@_stroke_block:nnnnn

```

#5 are the options

```

8108 { #5 } { #1 } { #2 } { #3 } { #4 }
8109 }
8110 \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
8111 { { #1 } { #2 } { #3 } { #4 } }
8112 }

8113 \clist_if_empty:NF \l_@@_borders_clist
8114 {
8115 \tl_gput_right:Nn \g_nicematrix_code_after_tl
8116 {
8117 \@@_stroke_borders_block:nnnnn
8118 { #5 } { #1 } { #2 } { #3 } { #4 }
8119 }
8120 }

```

```

8121 \tl_if_empty:NF \l_@@_fill_tl
8122 {
8123   \@@_add_opacity_to_fill:
8124   \tl_gput_right:Ne \g_@@_pre_code_before_tl
8125   {
8126     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
8127     { #1 - #2 }
8128     { #3 - #4 }
8129     { \dim_use:N \l_@@_rounded_corners_dim }
8130   }
8131 }
8132 \seq_if_empty:NF \l_@@_tikz_seq
8133 {
8134   \tl_gput_right:Ne \g_nicematrix_code_before_tl
8135   {
8136     \@@_block_tikz:nnnnn
8137     { \seq_use:Nn \l_@@_tikz_seq { , } }
8138     { #1 } { #2 } { #3 } { #4 }

```

We will have in that last field a list of lists of TikZ keys.

```

8139   }
8140 }
8141 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
8142 {
8143   \tl_gput_right:Ne \g_@@_rules_tl
8144   {
8145     \@@_draw_diagbox:nnnnnn
8146     { #1 } { #2 } { #3 } { #4 }
8147     { \exp_not:n { ##1 } }
8148     { \exp_not:n { ##2 } }
8149   }
8150 }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} &      & one      & \\\
                        &      & two      & \\\
three                  & four & five     & \\\
six                    & seven & eight    & \\\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block		one
		two
three	four	five
six	seven	eight

We highlight the node `1-1-block-short`

our block		one
		two
three	four	five
six	seven	eight

The construction of the node corresponding to the merged cells.

```

8151 \pgfpicture
8152 \pgfrememberpicturepositiononpagetrue
8153 \pgf@relevantforpicturesizefalse
8154 \@@_qpoint:n { row - #1 }
8155 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8156 \@@_qpoint:n { col - #2 }
8157 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8158 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }

```

```

8159 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8160 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8161 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x

```

We construct the node for the block with the name (#1-#2-block).

The function \@@_pgf_rect_node:nnnnn takes in as arguments the name of the node and the four coordinates of two opposite corner points of the rectangle.

```

8162 \@@_pgf_rect_node:nnnnn
8163 { \@@_env: - #1 - #2 - block }
8164 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8165 \str_if_empty:NF \l_@@_block_name_str
8166 {
8167   \pgfnodealias
8168   { \@@_env: - \l_@@_block_name_str }
8169   { \@@_env: - #1 - #2 - block }
8170   \str_if_empty:NF \l_@@_name_str
8171   {
8172     \pgfnodealias
8173     { \l_@@_name_str - \l_@@_block_name_str }
8174     { \@@_env: - #1 - #2 - block }
8175   }
8176 }

```

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.

```

8177 \bool_if:NF \l_@@_hpos_of_block_cap_bool
8178 {
8179   \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.

```

8180 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8181 {

```

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.

```

8182 \cs_if_exist:cT
8183 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8184 {
8185   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8186   {
8187     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
8188     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
8189   }
8190 }
8191 }

```

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.

```

8192 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
8193 {
8194   \@@_qpoint:n { col - #2 }
8195   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8196 }
8197 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
8198 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8199 {
8200   \cs_if_exist:cT
8201   { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8202   {
8203     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8204     {

```

```

8205         \pgfpointanchor
8206         { \@@_env: - #1 - \int_use:N \l_@@_last_col_int }
8207         { east }
8208         \dim_set:Nn \l_@@_tmpd_dim
8209         { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
8210     }
8211 }
8212 }
8213 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
8214 {
8215     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8216     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8217 }
8218 \@@_pgf_rect_node:nnnnn
8219 { \@@_env: - #1 - #2 - block - short }
8220 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8221 }

```

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.

```

8222 \bool_if:NT \l_@@_medium_nodes_bool
8223 {
8224     \@@_pgf_rect_node:nnn
8225     { \@@_env: - #1 - #2 - block - medium }
8226     { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
8227     {
8228         \pgfpointanchor
8229         { \@@_env:
8230             - \int_use:N \l_@@_last_row_int
8231             - \int_use:N \l_@@_last_col_int - medium
8232         }
8233         { south-east }
8234     }
8235 }
8236 \endpgfpicture
8237

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand `&`.

```

8238 \bool_if:NTF \l_@@_ampersand_bool
8239 {
8240     \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
8241     \int_zero_new:N \l_@@_split_int
8242     \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }

```

The following counters will be used to send information to `\cellcolor` if the user uses that command in a subcell. We use locally counters that have another signification in the main environment (maybe we should change that).

```

8243 \int_set:Nn \l_@@_first_row_int { #1 }
8244 \int_set:Nn \l_@@_first_col_int { #2 }
8245 \int_set:Nn \l_@@_last_row_int { #3 }
8246 \int_set:Nn \l_@@_last_col_int { #4 }
8247
8248 \pgfpicture
8249 \pgfrememberpicturepositiononpagetrue
8250 \pgf@relevantforpicturesizefalse
8251 \@@_qpoint:n { row - #1 }
8252 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8253 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8254 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
8255 \@@_qpoint:n { col - #2 }
8256 \dim_set_eq:NN \l_tmpa_dim \pgf@x
8257 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8258 \dim_set:Nn \l_tmpb_dim
8259 { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }

```

```

8259 \bool_lazy_or:nnT
8260 \l_@@_vlines_block_bool
8261 { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
8262 {
8263   \int_step_inline:nn { \l_@@_split_int - 1 }
8264   {
8265     \pgfpathmoveto
8266     {
8267       \pgfpoint
8268       { \l_tmpa_dim + ##1 \l_tmpb_dim }
8269       \l_@@_tmpc_dim
8270     }
8271     \pgfpathlineto
8272     {
8273       \pgfpoint
8274       { \l_tmpa_dim + ##1 \l_tmpb_dim }
8275       \l_@@_tmpd_dim
8276     }
8277     \CT@arc@
8278     \pgfsetlinewidth { 1.1 \arrayrulewidth }
8279     \pgfsetrectcap
8280     \pgfusepathqstroke
8281   }
8282 }
8283 \cs_set_eq:NN \cellcolor \@@_subcellcolor
8284 \int_zero_new:N \l_@@_split_i_int
8285 \str_if_eq:eeTF \l_@@_vpos_block_str T
8286 {
8287   \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
8288   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8289 }
8290 {
8291   \str_if_eq:eeTF \l_@@_vpos_block_str B
8292   {
8293     \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
8294     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8295   }
8296   {
8297     \bool_lazy_or:nnTF
8298     { \int_compare_p:nNn { #1 } = { #3 } }
8299     { \str_if_eq_p:ee \l_@@_vpos_block_str t }
8300     {
8301       \@@_qpoint:n { row - #1 - base }
8302       \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8303     }
8304     {
8305       \str_if_eq:eeTF \l_@@_vpos_block_str b
8306       {
8307         \@@_qpoint:n { row - #3 - base }
8308         \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8309       }
8310       {
8311         \@@_qpoint:n { #1 - #2 - block }
8312         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8313       }
8314     }
8315   }
8316 }
8317 \int_step_inline:nn \l_@@_split_int
8318 {
8319   \group_begin:

```

The counter `\l_@@_split_i_int` is only for the command `\@@_subcellcolor`.

```

8320 \int_set:Nn \l_@@_split_i_int { ##1 }
8321 \dim_set:Nn \col@sep
8322 { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
8323 \pgftransformshift
8324 {
8325   \pgfpoint
8326   {
8327     \l_tmpa_dim + ##1 \l_tmpb_dim -
8328     \str_case:on \l_@@_hpos_block_str
8329     {
8330       l { \l_tmpb_dim + \col@sep }
8331       c { 0.5 \l_tmpb_dim }
8332       r { \col@sep }
8333     }
8334   }
8335   { \l_@@_tmpc_dim }
8336 }
8337 \pgfset { inner~sep = \c_zero_dim }
8338 \pgfnode
8339 { rectangle }
8340 {
8341   \str_if_eq:eeTF T \l_@@_vpos_block_str
8342   {
8343     \str_case:on \l_@@_hpos_block_str
8344     {
8345       l { north~west }
8346       c { north }
8347       r { north~east }
8348     }
8349   }
8350   {
8351     \str_if_eq:eeTF B \l_@@_vpos_block_str
8352     {
8353       \str_case:on \l_@@_hpos_block_str
8354       {
8355         l { south~west }
8356         c { south }
8357         r { south~east }
8358       }
8359     }
8360     {
8361       \bool_lazy_any:nTF
8362       {
8363         { \int_compare_p:nNn { #1 } = { #3 } }
8364         { \str_if_eq_p:ee t \l_@@_vpos_block_str }
8365         { \str_if_eq_p:ee b \l_@@_vpos_block_str }
8366       }
8367       {
8368         \str_case:on \l_@@_hpos_block_str
8369         {
8370           l { base~west }
8371           c { base }
8372           r { base~east }
8373         }
8374       }
8375       {
8376         \str_case:on \l_@@_hpos_block_str
8377         {
8378           l { west }
8379           c { center }
8380           r { east }
8381         }
8382       }
8383     }
8384   }

```

```

8383         }
8384     }
8385 }
8386 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8387 \group_end:
8388 }
8389 \endpgfpicture
8390 }

```

Now the case where there is no ampersand & in the content of the block.

```

8391 {
8392     \bool_if:NTF \l_@@_p_block_bool
8393     {

```

When the final user has used the key p, we have to compute the width.

```

8394         \pgfpicture
8395         \pgfrememberpicturepositiononpagetrue
8396         \pgf@relevantforpicturesizefalse
8397         \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8398         {
8399             \@@_qpoint:n { col - #2 }
8400             \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8401             \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8402         }
8403         {
8404             \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8405             \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8406             \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8407         }
8408         \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8409         \endpgfpicture
8410         \hbox_set:Nn \l_@@_cell_box
8411         {
8412             \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8413                 { \g_tmpb_dim }
8414             \str_case:on \l_@@_hpos_block_str
8415                 { c \centering r \raggedleft l \raggedright j { } }
8416             #6
8417             \end { minipage }
8418         }
8419     }
8420     { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
8421     \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.

```

8422         \pgfpicture
8423         \pgfrememberpicturepositiononpagetrue
8424         \pgf@relevantforpicturesizefalse
8425         \bool_lazy_any:nTF
8426         {
8427             { \str_if_empty_p:N \l_@@_vpos_block_str }
8428             { \str_if_eq_p:ee c \l_@@_vpos_block_str }
8429             { \str_if_eq_p:ee T \l_@@_vpos_block_str }
8430             { \str_if_eq_p:ee B \l_@@_vpos_block_str }
8431         }
8432     {

```

If we are in the “first column”, we must put the block as if it was with the key r.

```

8433         \int_if_zero:nT { #2 } { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_r_str }

```

If we are in the “last column”, we must put the block as if it was with the key 1.

```

8434     \bool_if:nT \g_@@_last_col_found_bool
8435     {
8436         \int_compare:nNnT { #2 } = \g_@@_col_total_int
8437         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_1_str }
8438     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8439     \tl_set:Ne \l_tmpa_tl
8440     {
8441         \str_case:on \l_@@_vpos_block_str
8442         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8443         { } {
8444             \str_case:on \l_@@_hpos_block_str
8445             {
8446                 c { center }
8447                 l { west }
8448                 r { east }
8449                 j { center }
8450             }
8451         }
8452     c {
8453         \str_case:on \l_@@_hpos_block_str
8454         {
8455             c { center }
8456             l { west }
8457             r { east }
8458             j { center }
8459         }
8460     }
8461 }
8462 T {
8463     \str_case:on \l_@@_hpos_block_str
8464     {
8465         c { north }
8466         l { north-west }
8467         r { north-east }
8468         j { north }
8469     }
8470 }
8471 }
8472 B {
8473     \str_case:on \l_@@_hpos_block_str
8474     {
8475         c { south }
8476         l { south-west }
8477         r { south-east }
8478         j { south }
8479     }
8480 }
8481 }
8482 }
8483 }
8484 \pgftransformshift
8485 {
8486     \pgfpointanchor
8487     {
8488         \@@_env: - #1 - #2 - block
8489         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8490     }
8491     \l_tmpa_tl

```



```

8492     }
8493     \pgfset { inner~sep = \c_zero_dim }
8494     \pgfnode
8495     { rectangle }
8496     \l_tmpa_tl
8497     { \box_use_drop:N \l_@@_cell_box } { } { }
8498 }

```

End of the case when `\l_@@_vpos_block_str` is equal to c, T or B. Now, the other cases.

```

8499 {
8500     \pgfextracty \l_tmpa_dim
8501     {
8502         \@@_qpoint:n
8503         {
8504             row - \str_if_eq:eeTF b \l_@@_vpos_block_str { #3 } { #1 }
8505             - base
8506         }
8507     }
8508     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

8509     \pgfpointanchor
8510     {
8511         \@@_env: - #1 - #2 - block
8512         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8513     }
8514     {
8515         \str_case:on \l_@@_hpos_block_str
8516         {
8517             c { center }
8518             l { west }
8519             r { east }
8520             j { center }
8521         }
8522     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8523     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8524     \pgfset { inner~sep = \c_zero_dim }
8525     \pgfnode
8526     { rectangle }
8527     {
8528         \str_case:on \l_@@_hpos_block_str
8529         {
8530             c { base }
8531             l { base~west }
8532             r { base~east }
8533             j { base }
8534         }
8535     }
8536     { \box_use_drop:N \l_@@_cell_box } { } { }
8537 }
8538 \endpgfpicture
8539 }
8540 \group_end:
8541 }
8542 \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).

```

8543 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8544 {

```

```

8545 \tl_if_empty:NF \l_@@_opacity_tl
8546 {
8547   \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8548     {
8549       \tl_set:Ne \l_@@_fill_tl
8550       {
8551         [ opacity = \l_@@_opacity_tl ,
8552         \tl_tail:o \l_@@_fill_tl
8553       }
8554     }
8555     {
8556       \tl_set:Ne \l_@@_fill_tl
8557       { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8558     }
8559   }
8560 }

```

The first argument of `\@@_stroke_block:nnn` is a list of options for the rectangle that you will stroke. The other arguments are the position of the block: *imin*, *jmin*, *imax* and *jmax*.

```

8561 \cs_new_protected:Npn \@@_stroke_block:nnnnn #1 #2 #3 #4 #5
8562 {
8563   \group_begin:
8564   \tl_clear:N \l_@@_draw_tl
8565   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8566   \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8567   \tl_if_empty:NF \l_@@_draw_tl
8568   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8569     \tl_if_eq:NnTF \l_@@_draw_tl { default }
8570     \CT@arc@
8571     { \@@_color:o \l_@@_draw_tl }
8572   }

```

The following code can't be put just before the `\pgfusepath { stroke }` (it's too late).

```

8573 \pgfsetcornersarced
8574 { \pgfpoint \l_@@_rounded_corners_dim \l_@@_rounded_corners_dim }
8575 \int_compare:nNnF { #2 } > \c@iRow
8576 {
8577   \int_compare:nNnF { #3 } > \c@jCol
8578   {
8579     \@@_qpoint:n { row - #2 }
8580     \dim_set_eq:NN \l_tmpb_dim \pgf@y
8581     \@@_qpoint:n { col - #3 }
8582     \dim_set_eq:NN \l_tmpc_dim \pgf@x
8583     \@@_qpoint:n
8584     { row - \int_eval:n { 1 + \int_min:nn \c@iRow { #4 } } }
8585     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8586     \@@_qpoint:n
8587     { col - \int_eval:n { 1 + \int_min:nn \c@jCol { #5 } } }
8588     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8589     \pgfpathrectanglecorners
8590     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8591     { \pgfpoint \pgf@x \l_tmpa_dim }
8592     \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
8593     \pgfusepathqstroke
8594     { \pgfusepath { stroke } }
8595   }
8596 }
8597 \group_end:
8598 }

```

Here is the set of keys for the command `\@@_stroke_block:nnnnn`.

```

8599 \keys_define:nn { nicematrix / BlockStroke }
8600 {
8601   color .tl_set:N = \l_@@_draw_tl ,
8602   draw .code:n =
8603     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8604   draw .default:n = default ,
8605   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8606   rounded-corners .default:n = 4 pt ,
8607   rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The key `line-width` is now deprecated.

```

8608   line-width .dim_set:N = \l_@@_line_width_dim
8609 }

```

The command `\@@_vlines_block:nnnnn` is used only once: in `\@@_Block_v:nnnnnn` which actually draw the block.

The first argument of `\@@_vlines_block:nnn` is the width of the rules that we will draw. The other arguments are the position of the block: *imin*, *jmin*, *imax* and *jmax*.

```

8610 \cs_new_protected:Npn \@@_vlines_block:nnnnn #1 #2 #3 #4 #5
8611 {
8612   \group_begin:

```

We are actually during the execution of the `\g_nicematrix_code_after_tl`. In that context `\arrayrulewidth` is the width of standard lines (we are drawing standard rules because, up to now, there is no way to draw the internal lines of a Block with a style of TikZ).

```

8613   \dim_set:Nn \arrayrulewidth { #1 }

```

Below, you use `\@@_draw_vrule:n` to draw to rules and that command will use `\g_@@_pos_of_blocks_seq` in order to *not* draw the rules within the blocks. That's why we filter the list of blocks in order to discard the blocks which encompass the current block.

```

8614   \seq_set_filter:NnN \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
8615   {
8616     \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #2 }
8617     ||
8618     \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #3 }
8619     ||
8620     \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #4 }
8621     ||
8622     \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #5 }
8623   }
8624   \bool_if:NT \l_@@_fix_vertex_bool
8625   {
8626     \int_compare:nNnT { #3 } > 1
8627     {
8628       \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8629       {
8630         { 1 }
8631         { 1 }
8632         { \int_use:N \c@iRow }
8633         { \int_eval:n { #3 -1 } }
8634         { }
8635       }
8636     }
8637     \int_compare:nNnT { #5 } < \c@jCol
8638     {
8639       \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8640       {
8641         { 1 }
8642         { \int_eval:n { #5 + 1 } }
8643         { \int_use:N \c@iRow }
8644         { \int_use:N \c@jCol }

```

```

8645         { }
8646     }
8647 }
8648 }
8649 \int_set:Nn \l_@@_start_int { #2 }
8650 \int_set:Nn \l_@@_end_int { #4 }
8651 \int_step_inline:nnn { #3 } { #5 + 1 }
8652 {
8653     \int_set:Nn \l_@@_position_int { ##1 }
8654     \@@_draw_vrule:
8655 }
8656 \group_end:
8657 }

```

The command `\@@_hlines_block:nnnnn` is used only once: in `\@@_Block_v:nnnnnn` which actually draws the block.

```

8658 \cs_new_protected:Npn \@@_hlines_block:nnnnn #1 #2 #3 #4 #5
8659 {
8660     \group_begin:

```

We are actually during the execution of the `\g_nicematrix_code_after_tl`. In that context `\arrayrulewidth` is the width of standard lines (we are drawing standard rules because, up to now, there is no way to draw the internal lines of a Block with a style of TikZ).

```

8661     \dim_set:Nn \arrayrulewidth { #1 }

```

We filter the list of blocks `\g_@@_pos_of_blocks_seq` in order to discard the blocks which encompass the current block (elsewhere, of course, the rules would not be drawn).

```

8662     \seq_set_filter:Nn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
8663     {
8664         \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #2 }
8665         ||
8666         \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #3 }
8667         ||
8668         \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #4 }
8669         ||
8670         \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #5 }
8671     }
8672     \bool_if:NT \l_@@_fix_vertex_bool
8673     {
8674         \int_compare:nNnT { #2 } > 1
8675         {
8676             \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8677             {
8678                 { 1 }
8679                 { 1 }
8680                 { \int_eval:n { #2 - 1 } }
8681                 { \int_use:N \c@jCol }
8682                 { }
8683             }
8684         }
8685         \int_compare:nNnT { #4 } < \c@iRow
8686         {
8687             \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8688             {
8689                 { \int_eval:n { #4 + 1 } }
8690                 { 1 }
8691                 { \int_use:N \c@iRow }
8692                 { \int_use:N \c@jCol }
8693                 { }
8694             }
8695         }
8696     }

```

```

8697 \int_set:Nn \l_@@_start_int { #3 }
8698 \int_set:Nn \l_@@_end_int { #5 }
8699 \int_step_inline:nnn { #2 } { #4 + 1 }
8700 {
8701     \int_set:Nn \l_@@_position_int { ##1 }
8702     \@@_draw_hrulerule:
8703 }
8704 \group_end:
8705 }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke.

```

8706 \cs_new_protected:Npn \@@_stroke_borders_block:nnnnn #1 #2 #3 #4 #5
8707 {
8708     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8709     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8710     \dim_compare:NnTF \l_@@_rounded_corners_dim > \c_zero_dim
8711     { \@@_error:n { borders~forbidden } }
8712     {
8713         \tl_clear_new:N \l_@@_borders_tikz_tl
8714         \keys_set:no { nicematrix / OnlyForTikzInBorders } \l_@@_borders_clist
8715         \tl_set:Nn \l_@@_tmpc_tl { #2 }
8716         \tl_set:Nn \l_@@_tmpd_tl { #3 }
8717         \tl_set:Nn \l_@@_tmpa_tl { \int_eval:n { #4 + 1 } }
8718         \tl_set:Nn \l_@@_tmpb_tl { \int_eval:n { #5 + 1 } }
8719         \@@_stroke_borders_block_i:
8720     }
8721 }
8722 \AtBeginDocument
8723 {
8724     \cs_new_protected:Npe \@@_stroke_borders_block_i:
8725     {
8726         \c_@@_pgfortikzpicture_tl
8727         \@@_stroke_borders_block_ii:
8728         \c_@@_endpgfortikzpicture_tl
8729     }
8730 }
8731 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8732 {
8733     \pgfrememberpicturepositiononpagetrue
8734     \pgf@relevantforpicturesizefalse
8735     \CT@arc@
8736     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8737     \clist_if_in:NnT \l_@@_borders_clist { right }
8738     { \@@_stroke_vertical:n \l_@@_tmpb_tl }
8739     \clist_if_in:NnT \l_@@_borders_clist { left }
8740     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8741     \clist_if_in:NnT \l_@@_borders_clist { bottom }
8742     { \@@_stroke_horizontal:n \l_@@_tmpa_tl }
8743     \clist_if_in:NnT \l_@@_borders_clist { top }
8744     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8745 }
8746 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8747 {
8748     tikz .code:n =
8749         \cs_if_exist:NTF \tikzpicture
8750         { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8751         { \@@_error:n { tikz~in~borders~without~tikz } } ,
8752     tikz .value_required:n = true ,
8753     top .code:n = ,
8754     bottom .code:n = ,
8755     left .code:n = ,

```

```

8756     right .code:n = ,
8757     unknown .code:n = \@@_error:n { bad~border }
8758 }

```

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).

```

8759 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8760 {
8761   \@@_qpoint:n \l_@@_tmpc_tl
8762   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8763   \@@_qpoint:n \l_tmpa_tl
8764   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8765   \@@_qpoint:n { #1 }
8766   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8767   {
8768     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8769     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8770     \pgfusepathqstroke
8771   }
8772   {
8773     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8774     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8775   }
8776 }

```

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).

```

8777 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8778 {
8779   \@@_qpoint:n \l_@@_tmpd_tl
8780   \clist_if_in:NnTF \l_@@_borders_clist { left }
8781   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8782   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8783   \@@_qpoint:n \l_tmpb_tl
8784   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8785   \@@_qpoint:n { #1 }
8786   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8787   {
8788     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8789     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8790     \pgfusepathqstroke
8791   }
8792   {
8793     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8794     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8795   }
8796 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

8797 \keys_define:nn { nicematrix / BlockBorders }
8798 {
8799   borders .clist_set:N = \l_@@_borders_clist ,
8800   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8801   rounded-corners .default:n = 4 pt ,
8802   rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The following key is deprecated.

```

8803   line-width .dim_set:N = \l_@@_line_width_dim ,
8804 }

```

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.

```
8805 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8806 {
8807   \begin { tikzpicture }
8808   \@@_clip_with_rounded_corners:
```

We use `clist_map_inline:nn` because #5 is a list of lists.

```
8809   \clist_map_inline:nn { #1 }
8810   {
```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```
8811     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8812     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8813     (
8814       [
8815         xshift = \dim_use:N \l_@@_offset_dim ,
8816         yshift = - \dim_use:N \l_@@_offset_dim
8817       ]
8818       #2 -| #3
8819     )
8820     rectangle
8821     (
8822       [
8823         xshift = - \dim_use:N \l_@@_offset_dim ,
8824         yshift = \dim_use:N \l_@@_offset_dim
8825       ]
8826       \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8827     ) ;
8828   }
8829   \end { tikzpicture }
8830 }
8831 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

8832 \keys_define:nn { nicematrix / SpecialOffset }
8833 {
8834   offset .dim_set:N = \l_@@_offset_dim ,
8835 }
```

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.

```
8836 \cs_new_protected:Npn \@@_NullBlock:
8837 { \@@_collect_options:n { \@@_NullBlock_i: } }
8838 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8839 { }
```

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.

```
8840 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
8841 {
8842   \tl_gput_right:Ne \g_@@_pre_code_before_tl
8843   {
```

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`).

```
8844     \@@_subcellcolor:nnnnnnn
8845     {
8846       \tl_if_blank:nTF { #1 }
```

```

8847         { { \exp_not:n { #2 } } }
8848         { [ #1 ] { \exp_not:n { #2 } } }
8849     }
8850     { \int_use:N \l_@@_first_row_int } % first row of the block
8851     { \int_use:N \l_@@_first_col_int } % first column of the block
8852     { \int_use:N \l_@@_last_row_int } % last row of the block
8853     { \int_use:N \l_@@_last_col_int } % last column of the block
8854     { \int_use:N \l_@@_split_int }
8855     { \int_use:N \l_@@_split_i_int }
8856 }
8857 \ignorespaces
8858 }
8859 \cs_new_protected:Npn \@@_subcellcolor:nnnnnn #1 #2 #3 #4 #5 #6 #7
8860 {
8861     \@@_color_opacity: #1
8862     \pgfpicture
8863     \pgf@relevantforpicturesizefalse
8864     \@@_qpoint:n { col - #3 }
8865     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8866     \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
8867     \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
8868     \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
8869     \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
8870     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8871     \@@_qpoint:n { row - #2 }
8872     \pgfpathrectanglecorners
8873     { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } \l_@@_tmpc_dim }
8874     { \pgfpoint \l_tmpb_dim \pgf@y }
8875     \pgfusepathqfill
8876     \endpgfpicture
8877 }

```

27 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8878 \keys_define:nn { nicematrix / Auto }
8879 {
8880     columns-type .tl_set:N = \l_@@_columns_type_tl ,
8881     columns-type .value_required:n = true ,
8882     l .meta:n = { columns-type = l } ,
8883     r .meta:n = { columns-type = r } ,
8884     c .meta:n = { columns-type = c } ,
8885     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8886     delimiters / color .value_required:n = true ,
8887     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8888     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8889     delimiters .value_required:n = true ,
8890     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8891     rounded-corners .default:n = 4 pt
8892 }
8893 \NewDocumentCommand \AutoNiceMatrixWithDelims
8894 { m m 0 { } } > { \SplitArgument { 1 } { - } } m 0 { } m ! 0 { } }
8895 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
8896 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8897 {

```

The group is for the protection of the keys.

```

8898     \group_begin:

```



```

8899 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8900 \use:e
8901 {
8902   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8903   { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8904   [ \exp_not:o \l_tmpa_tl ]
8905 }
8906 \int_if_zero:nT \l_@@_first_row_int
8907 {
8908   \int_if_zero:nT \l_@@_first_col_int { & }
8909   \prg_replicate:nn { #4 - 1 } { & }
8910   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8911 }
8912 \prg_replicate:nn { #3 }
8913 {
8914   \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`).

```

8915   \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8916   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8917 }
8918 \int_compare:nNnT \l_@@_last_row_int > { -2 }
8919 {
8920   \int_if_zero:nT \l_@@_first_col_int { & }
8921   \prg_replicate:nn { #4 - 1 } { & }
8922   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8923 }
8924 \end { NiceArrayWithDelims }
8925 \group_end:
8926 }
8927 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
8928 {
8929   \cs_set_protected:cpn { #1 AutoNiceMatrix }
8930   {
8931     \bool_gset_true:N \g_@@_delims_bool
8932     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8933     \AutoNiceMatrixWithDelims { #2 } { #3 }
8934   }
8935 }

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8936 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
8937 {
8938   \group_begin:
8939   \bool_gset_false:N \g_@@_delims_bool
8940   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8941   \group_end:
8942 }

```

28 The redefinition of the command `\dotfill`

```

8943 \cs_set_eq:NN \@@_old_dotfill: \dotfill
8944 \cs_new_protected:Npn \@@_dotfill:
8945 {

```

First, we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in case of use of `\dotfill` “internally” in the cell (e.g. `\hbox to 1cm {\dotfill}`).

```

8946 \@@_old_dotfill:
8947 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8948 }

```

Now, if the box is not empty (unfortunately, we can't actually test whether the box is empty and that's why we only consider its 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`.

```

8949 \cs_new_protected:Npn \@@_dotfill_i:
8950 {
8951   \dim_compare:nNtT { \box_wd:N \l_@@_cell_box } = \c_zero_dim
8952   { \@@_old_dotfill: }
8953 }

```

29 The command `\diagbox`

The command `\diagbox` will be linked to `\diagbox:nn` in the environments of `nicematrix`. However, there are also redefinitions of `\diagbox` in other circumstances.

```

8954 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8955 {
8956   \tl_gput_right:Ne \g_@@_rules_tl
8957   {
8958     \@@_draw_diagbox:nnnnnn
8959     { \int_use:N \c@iRow }
8960     { \int_use:N \c@jCol }
8961     { \int_use:N \c@iRow }
8962     { \int_use:N \c@jCol }

```

The expansion done on `\g_@@_row_style_tl` will result in the fact that you take into account the current number of row and number of column.

```

8963     { \g_@@_row_style_tl \exp_not:n { #1 } }
8964     { \g_@@_row_style_tl \exp_not:n { #2 } }
8965   }

```

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`.

```

8966   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8967   {
8968     { \int_use:N \c@iRow }
8969     { \int_use:N \c@jCol }
8970     { \int_use:N \c@iRow }
8971     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8972     { }
8973   }
8974 }

```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_draw_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.

```

8975 \cs_new_protected:Npn \@@_draw_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8976 {
8977   \group_begin:
8978   \@@_qpoint:n { row - #1 }
8979   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8980   \@@_qpoint:n { col - #2 }
8981   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8982   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8983   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8984   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8985   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }

```

```

8986 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8987 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8988 {

```

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.

```

8989 \CT@arc@
8990 \pgfsetroundcap
8991 \pgfusepathqstroke
8992 }
8993 \pgfset { inner~sep = 1 pt }
8994 \pgfscope
8995 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8996 \pgfnode { rectangle } { south~west }
8997 {
8998 \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8999 \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
9000 \end { minipage }
9001 }
9002 { }
9003 { }
9004 \endpgfscope
9005 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
9006 \pgfnode { rectangle } { north~east }
9007 {
9008 \begin { minipage } { 20 cm }
9009 \raggedleft
9010 \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
9011 \end { minipage }
9012 }
9013 { }
9014 { }
9015 \group_end:
9016 }

```

30 The keyword `\CodeAfter`

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment `{@@-light-syntax}` on p. 88.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

9017 \cs_new:Npn \@@_CodeAfter: { \omit \@@_CodeAfter_ii:n }

```

However, in each cell of the environment, the command `\CodeAfter` will be linked to the following command `\@@_CodeAfter_ii:n` which begins with `\`.

```

9018 \cs_new_protected:Npn \@@_CodeAfter_i: { \ \omit \@@_CodeAfter_ii:n }

```

We have to catch everything until the end of the current environment (of `nicematrix`). First, we go until the next command `\end`.

```

9019 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
9020 {
9021 \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
9022 \@@_CodeAfter_iv:n
9023 }

```

We catch the argument of the command `\end` (in `#1`).

```
9024 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
9025 {
```

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.

```
9026 \str_if_eq:eeTF \currenvir { #1 }
9027 { \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`.

```
9028 {
9029 \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
9030 \@@_CodeAfter_ii:n
9031 }
9032 }
```

31 The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_pre_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of column. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```
9033 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
9034 {
9035 \pgfpicture
9036 \pgfrememberpicturepositiononpagetrue
9037 \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.

```
9038 \@@_qpoint:n { row - 1 }
9039 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
9040 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
9041 \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).

```
9042 \bool_if:nTF { #3 }
9043 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
9044 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
9045 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
9046 {
9047 \cs_if_exist:cT
9048 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
9049 {
9050 \pgfpointanchor
9051 { \@@_env: - ##1 - #2 }
9052 { \bool_if:nTF { #3 } { west } { east } }
9053 \dim_set:Nn \l_tmpa_dim
9054 {
9055 \bool_if:nTF { #3 }
9056 \dim_min:nn
```

```

9057         \dim_max:nn
9058         \l_tmpa_dim
9059         \pgf@x
9060     }
9061 }
9062 }

```

Now we can put the delimiter with a node of PGF.

```

9063 \pgfset { inner~sep = \c_zero_dim }
9064 \dim_zero:N \nulldelimiterspace
9065 \pgftransformshift
9066 {
9067     \pgfpoint
9068     \l_tmpa_dim
9069     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
9070 }
9071 \pgfnode
9072 { rectangle }
9073 { \bool_if:nTF { #3 } { east } { west } }
9074 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

9075     \nullfont
9076     $ % $
9077     \l_@@_color:o \l_@@_delimiters_color_tl
9078     \bool_if:nTF { #3 } { \left #1 } { \left . }
9079     \vcenter
9080     {
9081         \nullfont
9082         \hrule \@height
9083             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
9084             \@depth \c_zero_dim
9085             \@width \c_zero_dim
9086     }
9087     \bool_if:nTF { #3 } { \right . } { \right #1 }
9088     $ % $
9089 }
9090 { }
9091 { }
9092 \endpgfpicture
9093 }

```

32 The command `\SubMatrix`

```

9094 \keys_define:nn { nicematrix / sub-matrix }
9095 {
9096     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
9097     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
9098     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
9099     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
9100     xshift .value_required:n = true ,
9101     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9102     delimiters / color .value_required:n = true ,
9103     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
9104     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9105     hlines .default:n = all ,
9106     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9107     vlines .default:n = all ,
9108     hvlines .meta:n = { hlines, vlines } ,
9109     hvlines .value_forbidden:n = true
9110 }

```

```

9111 \keys_define:nn { nicematrix }
9112 {
9113   SubMatrix .inherit:n = nicematrix / sub-matrix ,
9114   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9115   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9116   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9117 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

9118 \keys_define:nn { nicematrix / SubMatrix }
9119 {
9120   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9121   delimiters / color .value_required:n = true ,
9122   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9123   hlines .default:n = all ,
9124   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9125   vlines .default:n = all ,
9126   hvlines .meta:n = { hlines, vlines } ,
9127   hvlines .value_forbidden:n = true ,
9128   name .code:n =
9129     \tl_if_empty:nTF { #1 }
9130     { \@@_error:n { Invalid-name } }
9131     {
9132       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
9133       {
9134         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
9135         { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
9136         {
9137           \str_set:Nn \l_@@_submatrix_name_str { #1 }
9138           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
9139         }
9140       }
9141       { \@@_error:n { Invalid-name } }
9142     } ,
9143   name .value_required:n = true ,
9144   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
9145   rules .value_required:n = true ,
9146   code .tl_set:N = \l_@@_code_tl ,
9147   code .value_required:n = true ,
9148   unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
9149 }

9150 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
9151 {
9152   \tl_gput_right:Ne \g_@@_pre_code_after_tl
9153   {
9154     \SubMatrix { #1 } { #2 } { #3 } { #4 }
9155     [
9156       delimiters / color = \l_@@_delimiters_color_tl ,
9157       hlines = \l_@@_submatrix_hlines_clist ,
9158       vlines = \l_@@_submatrix_vlines_clist ,
9159       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
9160       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
9161       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
9162       slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
9163       #5
9164     ]
9165   }
9166   \@@_SubMatrix_in_code_before_i { #2 } { #3 }
9167   \ignorespaces
9168 }

9169 \NewDocumentCommand \@@_SubMatrix_in_code_before_i

```

```

9170 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9171 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

9172 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
9173 {
9174   \seq_gput_right:Ne \g_@@_submatrix_seq
9175   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

9176   { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
9177   { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
9178   { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
9179   { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
9180   }
9181 }

```

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).

```

9182 \NewDocumentCommand \@@_compute_i_j:nn
9183 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9184 { \@@_compute_i_j:nnnn #1 #2 }

9185 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
9186 {
9187   \def \l_@@_first_i_tl { #1 }
9188   \def \l_@@_first_j_tl { #2 }
9189   \def \l_@@_last_i_tl { #3 }
9190   \def \l_@@_last_j_tl { #4 }
9191   \tl_if_eq:NnT \l_@@_first_i_tl { last }
9192     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
9193   \tl_if_eq:NnT \l_@@_first_j_tl { last }
9194     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
9195   \tl_if_eq:NnT \l_@@_last_i_tl { last }
9196     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
9197   \tl_if_eq:NnT \l_@@_last_j_tl { last }
9198     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
9199 }

```

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`.

```

9200 \AtBeginDocument
9201 {
9202   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
9203   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
9204     { \@@_sub_matrix:nnnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
9205 }

```

```

9206 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
9207 {
9208   \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

9209   \@@_compute_i_j:nn { #2 } { #3 }
9210   \int_compare:nNt \l_@@_first_i_tl = \l_@@_last_i_tl
9211     { \def \arraystretch { 1 } }
9212   \bool_lazy_or:nnTF
9213     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9214     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9215     { \@@_error:nn { Construct~too~large } { \SubMatrix } }
9216   {
9217     \str_clear_new:N \l_@@_submatrix_name_str
9218     \keys_set:nn { nicematrix / SubMatrix } { #5 }
9219     \pgfpicture
9220     \pgfrememberpicturepositiononpagetrue
9221     \pgf@relevantforpicturesizefalse
9222     \pgfset { inner~sep = \c_zero_dim }
9223     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9224     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int_step_inline:nnn is provided by curryfication.

```

9225     \bool_if:NTF \l_@@_submatrix_slim_bool
9226     { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
9227     { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
9228     {
9229       \cs_if_exist:cT
9230       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9231       {
9232         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9233         \dim_compare:nNt \pgf@x < \l_@@_x_initial_dim
9234         { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9235       }
9236       \cs_if_exist:cT
9237       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9238       {
9239         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9240         \dim_compare:nNt \pgf@x > \l_@@_x_final_dim
9241         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9242       }
9243     }
9244     \dim_compare:nNtTF \l_@@_x_initial_dim = \c_max_dim
9245     { \@@_error:nn { Impossible~delimiter } { left } }
9246     {
9247       \dim_compare:nNtTF \l_@@_x_final_dim = { - \c_max_dim }
9248       { \@@_error:nn { Impossible~delimiter } { right } }
9249       { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
9250     }
9251   \endpgfpicture
9252 }
9253 \group_end:
9254 \ignorespaces
9255 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

9256 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
9257 {
9258   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
9259   \dim_set:Nn \l_@@_y_initial_dim
9260   {
9261     \fp_to_dim:n
9262     {
9263       \pgf@y

```



```

9264         + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
9265     }
9266 }
9267 \@@_qpoint:n { row - \l_@@_last_i_tl - base }
9268 \dim_set:Nn \l_@@_y_final_dim
9269 { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
9270 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
9271 {
9272     \cs_if_exist:cT
9273     { \pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
9274     {
9275         \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
9276         \dim_set:Nn \l_@@_y_initial_dim
9277         { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
9278     }
9279     \cs_if_exist:cT
9280     { \pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
9281     {
9282         \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
9283         \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
9284         { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
9285     }
9286 }
9287 \dim_set:Nn \l_tmpa_dim
9288 {
9289     \l_@@_y_initial_dim - \l_@@_y_final_dim +
9290     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
9291 }
9292 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

9293 \group_begin:
9294 \pgfsetlinewidth { 1.1 \arrayrulewidth }
9295 \@@_set_CTarc:o \l_@@_rules_color_tl
9296 \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`.

```

9297 \seq_map_inline:Nn \g_@@_cols_vlism_seq
9298 {
9299     \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
9300     {
9301         \int_compare:nNnT
9302         { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
9303         {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

9304         \@@_qpoint:n { col - ##1 }
9305         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9306         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9307         \pgfusepathqstroke
9308     }
9309 }
9310 }

```

Now, we draw the vertical rules specified in the key `vlines` of `\SubMatrix`. The last argument of `\int_step_inline:nn` or `\clist_map_inline:Nn` is given by curryfication.

```

9311 \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
9312 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
9313 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
9314 {
9315     \bool_lazy_and:nnTF

```

```

9316 { \int_compare_p:nNn { ##1 } > \c_zero_int }
9317 {
9318   \int_compare_p:nNn
9319     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
9320 {
9321   \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
9322   \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9323   \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9324   \pgfusepathqstroke
9325 }
9326 { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
9327 }

```

Now, we draw the horizontal rules specified in the key `hlines` of `\SubMatrix`. The last argument of `\int_step_inline:nn` or `\clist_map_inline:Nn` is given by curryfication.

```

9328 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
9329 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
9330 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
9331 {
9332   \bool_lazy_and:nnTF
9333     { \int_compare_p:nNn { ##1 } > \c_zero_int }
9334     {
9335       \int_compare_p:nNn
9336         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
9337     {
9338       \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

9339   \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

9340   \dim_set:Nn \l_tmpa_dim
9341     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9342   \str_case:nn { #1 }
9343   {
9344     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9345     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
9346     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9347   }
9348   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

9349   \dim_set:Nn \l_tmpb_dim
9350     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9351   \str_case:nn { #2 }
9352   {
9353     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9354     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
9355     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9356   }
9357   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9358   \pgfusepathqstroke
9359   \group_end:
9360 }
9361 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
9362 }

```

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).

```

9363   \str_if_empty:NF \l_@@_submatrix_name_str
9364   {
9365     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
9366     \l_@@_x_initial_dim \l_@@_y_initial_dim
9367     \l_@@_x_final_dim \l_@@_y_final_dim

```

```

9368     }
9369     \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`.

```

9370     \begin { pgfscope }
9371     \pgftransformshift
9372     {
9373         \pgfpoint
9374         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9375         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9376     }
9377     \str_if_empty:NTF \l_@@_submatrix_name_str
9378     { \@@_node_left:nn #1 { } }
9379     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9380     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9381     \pgftransformshift
9382     {
9383         \pgfpoint
9384         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9385         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9386     }
9387     \str_if_empty:NTF \l_@@_submatrix_name_str
9388     { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9389     {
9390         \@@_node_right:nnnn #2
9391         { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9392     }

```

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`.

```

9393     \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9394     \flag_clear_new:N \l_@@_code_flag
9395     \l_@@_code_tl
9396 }

```

In the key code of the command `\SubMatrix` there may be TikZ instructions. We want that, in these instructions, the i and j in specifications of nodes of the forms $i-j$, $\text{row-}i$, $\text{col-}j$ and $i-|j$ refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

9397 \cs_set_eq:NN \@@_old_pgfpointanchor: \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument `#1` of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of TikZ nodes which are computed in an expandable way.

The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by currying.

```

9398 \cs_new:Npn \@@_pgfpointanchor:n #1
9399 { \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`).

```

9400 \cs_new:Npn \@@_pgfpointanchor_i:n #1
9401 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }
9402 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9403 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9404 \str_if_empty:nTF { #1 }

```

First, when the name of the name begins with `\tikz@pp@name`.

```

9405 { \@@_pgfpointanchor_iv:w #2 }

```

And now, when there is no `\tikz@pp@name`.

```

9406 { \@@_pgfpointanchor_ii:n { #1 } }
9407 }

```

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`).

```

9408 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
9409 { \@@_pgfpointanchor_ii:n { #1 } }

```

With the command `\@@_pgfpointanchor_ii:n`, we deal with the actual name of the node (without the `\tikz@pp@name`). First, we have to detect whether it is of the form `i` of the form `i-j` (with an hyphen).

Remark: It would be possible to test the presence of the hyphen in an expandable way to using `\etl_if_in:nNTF` of the package `etl` but, as of now, we do not load `etl`.

```

9410 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

```

```

9411 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
9412 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9413 \str_if_empty:nTF { #2 }

```

First the case where the argument does *not* contain an hyphen.

```

9414 { \@@_pgfpointanchor_iii:n { #1 } }

```

And now the case the argument contains a hyphen. In that case, we have a weird construction because we must retrieve the extra hyphen we have added as marker (cf. `\@@_pgfpointanchor_ii:n`).

```

9415 { \@@_pgfpointanchor_iii:w { #1 } #2 }
9416 }

```

The following function is for the case when the name contains an hyphen.

```

9417 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9418 {

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9419 \@@_env:
9420 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9421 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9422 }

```

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.

```

9423 \tl_const:Nn \c_@@_integers_alist_tl
9424 {
9425 { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9426 { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9427 { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9428 { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9429 }

```

```

9430 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9431 {

```

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`.

```

9432 \str_case:nVTF { #1 } \c_@@_integers_alist_tl
9433 {
9434 \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9435 \@@_env: -
9436 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9437 { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9438 { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9439 }
9440 {
9441 \str_if_eq:eeTF { #1 } { last }
9442 {
9443 \flag_raise:N \l_@@_code_flag
9444 \@@_env: -
9445 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9446 { \int_eval:n { \l_@@_last_i_tl + 1 } }
9447 { \int_eval:n { \l_@@_last_j_tl + 1 } }
9448 }
9449 { #1 }
9450 }
9451 }

```

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`).

```

9452 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9453 {
9454 \pgfnode
9455 { rectangle }
9456 { east }
9457 {
9458 \nullfont
9459 $ % $
9460 \@@_color:o \l_@@_delimiters_color_tl
9461 \left #1
9462 \vcenter
9463 {
9464 \nullfont
9465 \hrule \@height \l_tmpa_dim
9466 \@depth \c_zero_dim
9467 \@width \c_zero_dim
9468 }
9469 \right .
9470 $ % $
9471 }
9472 { #2 }
9473 { }
9474 }

```

The command `\@@_node_right:nn` puts the right delimiter with the correct size. The argument #1 is the delimiter to put. The argument #2 is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`). The argument #3 is the subscript and #4 is the superscript.

```

9475 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9476 {
9477   \pgfnode
9478   { rectangle }
9479   { west }
9480   {
9481     \nullfont
9482     $ % $
9483     \colorlet { current-color } { . }
9484     \@@_color:o \l_@@_delimiters_color_tl
9485     \left .
9486     \vcenter
9487     {
9488       \nullfont
9489       \hrule \@height \l_tmpa_dim
9490               \@depth \c_zero_dim
9491               \@width \c_zero_dim
9492     }
9493     \right #1
9494     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9495     ~ { \color { current-color } \smash { #4 } }
9496     $ % $
9497   }
9498   { #2 }
9499   { }
9500 }

```

33 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9501 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9502 {
9503   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9504   \ignorespaces
9505 }
9506 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9507 {
9508   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9509   \ignorespaces
9510 }
9511 \keys_define:nn { nicematrix / Brace }
9512 {
9513   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9514   left-shorten .value_forbidden:n = true ,
9515   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9516   right-shorten .value_forbidden:n = true ,
9517   shorten .meta:n = { left-shorten , right-shorten } ,
9518   shorten .value_forbidden:n = true ,
9519   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9520   color .tl_set:N = \l_tmpa_tl ,
9521   color .value_required:n = true ,
9522   unknown .code:n =
9523     \@@_unknown_key:nn
9524     { nicematrix / Brace }

```

```

9525     { Unknown-key-for-Brace }
9526 }

```

#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`.

```

9527 \cs_new_protected:Npn \l_@@_brace:nnnnn #1 #2 #3 #4 #5
9528 {
9529   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9530   \l_@@_compute_i_j:nn { #1 } { #2 }
9531   \bool_lazy_or:nnTF
9532     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9533     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9534   {
9535     \str_if_eq:eeTF { #5 } { under }
9536     { \l_@@_error:nn { Construct-too-large } { \UnderBrace } }
9537     { \l_@@_error:nn { Construct-too-large } { \OverBrace } }
9538   }
9539   {
9540     \tl_clear:N \l_tmpa_tl
9541     \keys_set:nn { nicematrix / Brace } { #4 }
9542     \tl_if_empty:NF \l_tmpa_tl { \color \l_tmpa_tl }
9543     \pgfpicture
9544     \pgfrememberpicturepositiononpagetrue
9545     \pgf@relevantforpicturesizefalse
9546     \bool_if:NT \l_@@_brace_left_shorten_bool
9547     {
9548       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9549       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9550       {
9551         \cs_if_exist:cT
9552           { pgf @ sh @ ns @ \l_@@_env: - ##1 - \l_@@_first_j_tl }
9553         {
9554           \pgfpointanchor { \l_@@_env: - ##1 - \l_@@_first_j_tl } { west }
9555
9556           \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9557             { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9558         }
9559       }
9560     }
9561     \bool_lazy_or:nnT
9562       { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9563       { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
9564     {
9565       \l_@@_qpoint:n { col - \l_@@_first_j_tl }
9566       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9567     }
9568     \bool_if:NT \l_@@_brace_right_shorten_bool
9569     {
9570       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9571       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9572       {
9573         \cs_if_exist:cT
9574           { pgf @ sh @ ns @ \l_@@_env: - ##1 - \l_@@_last_j_tl }
9575         {
9576           \pgfpointanchor { \l_@@_env: - ##1 - \l_@@_last_j_tl } { east }
9577           \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9578             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9579         }
9580       }
9581     }
9582     \bool_lazy_or:nnT

```

```

9583     { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9584     { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
9585     {
9586         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9587         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9588     }
9589     \pgfset { inner~sep = \c_zero_dim }
9590     \str_if_eq:eeTF { #5 } { under }
9591     { \@@_underbrace_i:n { #3 } }
9592     { \@@_overbrace_i:n { #3 } }
9593     \endpgfpicture
9594 }
9595 \group_end:
9596 }

```

The argument is the text to put above the brace.

```

9597 \cs_new_protected:Npn \@@_overbrace_i:n #1
9598 {
9599     \@@_qpoint:n { row - \l_@@_first_i_tl }
9600     \pgftransformshift
9601     {
9602         \pgfpoint
9603         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9604         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9605     }
9606     \pgfnode
9607     { rectangle }
9608     { south }
9609     {
9610         \vtop
9611         {
9612             \group_begin:
9613             \everycr { }
9614             \halign
9615             {
9616                 \hfil ## \hfil \crcr
9617                 \bool_if:NTF \l_@@_tabular_bool
9618                 { \begin { tabular } { c } #1 \end { tabular } }
9619                 { $ \begin { array } { c } #1 \end { array } $ }
9620                 \cr
9621                 $ % $
9622                 \overbrace
9623                 {
9624                     \hbox_to_wd:nn
9625                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9626                     { }
9627                 }
9628                 $ % $
9629                 \cr
9630             }
9631             \group_end:
9632         }
9633     }
9634     { }
9635     { }
9636 }

```

The argument is the text to put under the brace.

```

9637 \cs_new_protected:Npn \@@_underbrace_i:n #1
9638 {
9639     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9640     \pgftransformshift
9641     {

```



```

9642     \pgfpoint
9643     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9644     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9645 }
9646 \pgfnode
9647 { rectangle }
9648 { north }
9649 {
9650     \group_begin:
9651     \everycr { }
9652     \vbox
9653     {
9654         \halign
9655         {
9656             \hfil ## \hfil \crcr
9657             $ % $
9658             \underbrace
9659             {
9660                 \hbox_to_wd:nn
9661                 { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9662                 { }
9663             }
9664             $ % $
9665             \cr
9666             \bool_if:NTF \l_@@_tabular_bool
9667             { \begin { tabular } { c } #1 \end { tabular } }
9668             { $ \begin { array } { c } #1 \end { array } $ }
9669             \cr
9670         }
9671     }
9672     \group_end:
9673 }
9674 { }
9675 { }
9676 }

```

34 The commands HBrace et VBrace

The TikZ style `nicematrix/brace` is a TikZ style used to draw the braces created by `\Hbrace` and `\Vbrace`.

We can't load that definition right away because of course, maybe the final user has not yet loaded TikZ (`\Hbrace` and `\Vbrace` are available only when TikZ is loaded and also its library `decorations.pathreplacing`).

```

9677 \AddToHook { package / tikz / after }
9678 {
9679     \tikzset
9680     {
9681         nicematrix / brace / .style =
9682         {
9683             decoration = { brace , raise = -0.15 em } ,
9684             decorate ,
9685         } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9686         nicematrix / mirrored-brace / .style =
9687         {
9688             nicematrix / brace ,
9689             decoration = mirror ,

```

```

9690     }
9691   }
9692 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9693 \keys_define:nn { nicematrix / Hbrace }
9694 {
9695   color .code:n = ,
9696   horizontal-label .code:n = ,
9697   horizontal-labels .code:n = ,
9698   shorten .code:n = ,
9699   shorten-start .code:n = ,
9700   shorten-end .code:n = ,
9701   shorten+ .code:n = ,
9702   shorten-start+ .code:n = ,
9703   shorten-end+ .code:n = ,
9704   shorten~+ .code:n = ,
9705   shorten-start~+ .code:n = ,
9706   shorten-end~+ .code:n = ,
9707   brace-shift .code:n = ,
9708   brace-shift+ .code:n = ,
9709   brace-shift~+ .code:n = ,
9710   unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9711 }

```

Here we need an “fully expandable” command.

```

9712 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9713 {
9714   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9715     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9716     { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9717 }

```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

9718 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9719 {
9720   \int_compare:nNnTF \c@iRow < { 2 }
9721   {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9722     \str_if_eq:nnTF { #2 } { * }
9723     {
9724       \bool_set_true:N \l_@@_nullify_dots_bool
9725       \Ldots
9726       [
9727         line-style = nicematrix / brace ,
9728         #1 ,
9729         up =
9730         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9731       ]
9732     }
9733     {
9734       \Hdotsfor
9735       [
9736         line-style = nicematrix / brace ,
9737         #1 ,
9738         up =
9739         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9740       ]
9741       { #2 }

```

```

9742     }
9743   }
9744   {
9745     \str_if_eq:nnTF { #2 } { * }
9746     {
9747       \bool_set_true:N \l_@@_nullify_dots_bool
9748       \Ldots
9749       [
9750         line-style = nicematrix / mirrored-brace ,
9751         #1 ,
9752         down =
9753           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9754       ]
9755     }
9756     {
9757       \Hdotsfor
9758       [
9759         line-style = nicematrix / mirrored-brace ,
9760         #1 ,
9761         down =
9762           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9763       ]
9764       { #2 }
9765     }
9766   }
9767   \keys_set:nn { nicematrix / Hbrace } { #1 }
9768 }

```

```

9769 \NewDocumentCommand { \@@_Vbrace } { 0 { } m m }
9770 {
9771   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9772   { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9773   { \@@_error:nn { Hbrace-not-allowed } { \Vbrace } }
9774 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

9775 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
9776 {
9777   \int_compare:nNnTF \c@jCol < { 2 }
9778   {
9779     \str_if_eq:nnTF { #2 } { * }
9780     {
9781       \bool_set_true:N \l_@@_nullify_dots_bool
9782       \Vdots
9783       [
9784         Vbrace ,
9785         line-style = nicematrix / mirrored-brace ,
9786         #1 ,
9787         down =
9788           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9789       ]
9790     }
9791     {
9792       \Vdotsfor
9793       [
9794         Vbrace ,
9795         line-style = nicematrix / mirrored-brace ,
9796         #1 ,
9797         down =
9798           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9799       ]
9800       { #2 }

```

```

9801     }
9802   }
9803   {
9804     \str_if_eq:nnTF { #2 } { * }
9805     {
9806       \bool_set_true:N \l_@@_nullify_dots_bool
9807       \Vdots
9808       [
9809         Vbrace ,
9810         line-style = nicematrix / brace ,
9811         #1 ,
9812         up =
9813         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9814       ]
9815     }
9816     {
9817       \Vdotsfor
9818       [
9819         Vbrace ,
9820         line-style = nicematrix / brace ,
9821         #1 ,
9822         up =
9823         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9824       ]
9825       { #2 }
9826     }
9827   }
9828   \keys_set:nn { nicematrix / Hbrace } { #1 }
9829 }

```

35 The command TikzEveryCell

```

9830 \bool_new:N \l_@@_not_empty_bool
9831 \bool_new:N \l_@@_empty_bool
9832
9833 \keys_define:nn { nicematrix / TikzEveryCell }
9834 {
9835   not-empty .code:n =
9836     \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
9837     { \bool_set_true:N \l_@@_not_empty_bool }
9838     { \@@_error:n { detection-of-empty-cells } } ,
9839   not-empty .value_forbidden:n = true ,
9840   empty .code:n =
9841     \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
9842     { \bool_set_true:N \l_@@_empty_bool }
9843     { \@@_error:n { detection-of-empty-cells } } ,
9844   empty .value_forbidden:n = true ,
9845   unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
9846 }
9847
9848
9849 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9850 {
9851   \IfPackageLoadedTF { tikz }
9852   {
9853     \group_begin:
9854     \keys_set:nn { nicematrix / TikzEveryCell } { #1 }

```

The inner pair of braces in the following line is mandatory because, the last argument of `\@@_tikz:nnnn` is a *list of lists* of TikZ keys.

```

9855 \tl_set:Nn \l_tmpa_tl { { #2 } }
9856 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9857 { \@@_for_a_block:nnnnn #1 }
9858 \@@_all_the_cells:
9859 \group_end:
9860 }
9861 { \@@_error:n { TikzEveryCell~without~tikz } }
9862 }
9863
9864
9865 \cs_new_protected:Nn \@@_all_the_cells:
9866 {
9867 \int_step_inline:nn \c@iRow
9868 {
9869 \int_step_inline:nn \c@jCol
9870 {
9871 \cs_if_exist:cF { cell - ##1 - #####1 }
9872 {
9873 \clist_if_in:NcF \l_@@_corners_cells_clist
9874 { ##1 - #####1 }
9875 {
9876 \bool_set_false:N \l_tmpa_bool
9877 \cs_if_exist:cTF
9878 { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
9879 {
9880 \bool_if:NF \l_@@_empty_bool
9881 { \bool_set_true:N \l_tmpa_bool }
9882 }
9883 {
9884 \bool_if:NF \l_@@_not_empty_bool
9885 { \bool_set_true:N \l_tmpa_bool }
9886 }
9887 \bool_if:NT \l_tmpa_bool
9888 {
9889 \@@_block_tikz:nnnnn
9890 \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
9891 }
9892 }
9893 }
9894 }
9895 }
9896 }
9897
9898 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9899 {
9900 \bool_if:NF \l_@@_empty_bool
9901 {
9902 \@@_block_tikz:nnnnn
9903 \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9904 }
9905 \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9906 }
9907
9908 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9909 {
9910 \int_step_inline:nnn { #1 } { #3 }
9911 {
9912 \int_step_inline:nnn { #2 } { #4 }
9913 { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9914 }
9915 }

```

36 The key draw-trees-in-col

```

9916 \cs_new_protected:Npn \@@_draw_trees:
9917 {
9918   \@@_expand_clist_hvlines:NN \g_@@_col_with_trees_clist \c@jCol
9919   \dim_zero_new:N \l_@@_em_dim
9920   \dim_set:Nn \l_@@_em_dim { 1 em }
9921   \dim_zero_new:N \l_@@_ex_dim
9922   \dim_set:Nn \l_@@_ex_dim { 1 ex }
9923   \pgfpicture
9924   \pgfrememberpicturepositiononpagetrue
9925   \pgf@relevantforpicturesizefalse
9926   \dim_compare:nNnT \l_@@_trees_line_width_dim > \c_zero_dim
9927     { \pgfsetlinewidth { \l_@@_trees_line_width_dim } }
9928   \@@_color:o \l_@@_trees_color_tl
9929   \pgfsetcornersarced
9930     { \pgfpoint \l_@@_trees_rounded_corners_dim \l_@@_trees_rounded_corners_dim }
9931   \clist_map_function:NN \g_@@_col_with_trees_clist
9932     \@@_draw_trees_in_col:n
9933   \clist_gclear:N \g_@@_col_with_trees_clist
9934   \endpgfpicture
9935 }

9936 \cs_new_protected:Npn \@@_draw_trees_in_col:n #1
9937 {

```

The argument is provided by curryfication.

```

9938   \int_compare:nNnTF { #1 } > \c@jCol
9939     { \@@_error:nn { Col~outside~tabular~in~trees } }
9940     {
9941       \int_compare:nNnTF { #1 } = \c@jCol
9942         { \@@_error:nn { Last~col~in~trees } }
9943         \@@_draw_trees_in_col_i:n
9944     }
9945   { #1 }
9946 }

9947 \cs_new_protected:Npn \@@_draw_trees_in_col_i:n #1
9948 {
9949   \int_set:Nn \l_tmpa_int { 1 }
9950   \int_step_inline:nn { \c@iRow + 1 }
9951   {
9952     \cs_if_exist:cT
9953       { pgf @ sh @ ns @ \@@_env: - ##1 - #1 }
9954       {
9955         \int_compare:nNnT { ##1 } > { \l_tmpa_int + 1 }
9956         {

```

Now, you will, potentially, draw a tree.

```

9957           \@@_draw_tree:nee
9958           { #1 }
9959           { \int_use:N \l_tmpa_int }
9960           { \int_eval:n { ##1 - 1 } } }
9961   }
9962   \int_set:Nn \l_tmpa_int { ##1 }
9963 }
9964 }
9965 \int_compare:nNnT \c@iRow > \l_tmpa_int
9966 {
9967   \@@_draw_tree:nee
9968   { #1 }
9969   { \int_use:N \l_tmpa_int }
9970   { \int_eval:n { \c@iRow } }

```

```

9971     }
9972 }

```

#1 is the number of column; #2 is the first row : the root of the tree; #3 is the last row of the blank zone where we will draw our tree.

```

9973 \cs_new_protected:Npn \@@_draw_tree:nnn #1 #2 #3
9974 {

```

\l_tmpa_dim will be the x -value of the vertical rule that we will draw.

```

9975     \pgfpointanchor { \@@_env: - #1 } { 5 }
9976     \dim_set_eq:NN \l_tmpa_dim \pgf@x

```

We will begin by the *last* branch of the tree. When that last branch has been drawn (with the vertical line), we will rise the boolean \l_tmpa_bool.

```

9977     \bool_set_false:N \l_tmpa_bool
9978     \int_step_inline:nnnn { #3 } { -1 } { #2 + 1 }
9979     {
9980         \cs_if_exist:cT
9981         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #1 + 1 } }

```

We have found the last branch to draw.

```

9982     {
9983         \pgfpointanchor
9984         { \@@_env: - ##1 - \int_eval:n { #1 + 1 } }
9985         { base-west }
9986         \pgfpathmoveto
9987         {
9988             \pgfpoint
9989             { \dim_eval:n { \pgf@x - 0.7 \l_@@_ex_dim } }
9990             { \dim_eval:n { \pgf@y + 0.25 \l_@@_em_dim } }
9991         }
9992         \pgfpathlineto { \pgfpoint \l_tmpa_dim \pgf@y }
9993         \bool_if:NF \l_tmpa_bool
9994         {
9995             \pgfpointanchor{ \@@_env: - #2 - #1 } { south }
9996             \pgfpathlineto
9997             { \pgfpoint \l_tmpa_dim { \dim_eval:n { \pgf@y - 3pt } } }
9998             \bool_set_true:N \l_tmpa_bool
9999         }
10000         \pgfusepath { stroke }
10001     }
10002 }
10003 }
10004 \cs_generate_variant:Nn \@@_draw_tree:nnn { n e e }

```

37 The key create-blocks-in-col

```

10005 \cs_new_protected:Npn \@@_create_blocks_in_col:
10006 {
10007     \@@_expand_clist_hvlines:NN \g_@@_cbic_clist \c@jCol
10008     \clist_map_inline:Nn \g_@@_cbic_clist
10009     {
10010         \cs_set:cpn
10011         {
10012             pgf @ sh @ ns @ \@@_env:
10013             - \int_eval:n { \c@iRow + 1 } - ##1 }
10014         { rien }

```

\l_tmpa_int will be the first row of the block being created.

```

10015     \int_set:Nn \l_tmpa_int { 1 }
10016     \int_step_inline:nn { \c@iRow + 1 }
10017     {
10018         \cs_if_exist:cT
10019         { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 }

```

```

10020         {
10021             \int_compare:nNnT { #####1 } > { \l_tmpa_int + 1 }
10022             {
10023                 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
10024                 {
10025                     { \int_use:N \l_tmpa_int }
10026                     { ##1 }
10027                     { \int_eval:n { #####1 - 1 } }
10028                     { ##1 }
10029                     { }
10030                 }
10031             }
10032             \int_set:Nn \l_tmpa_int { #####1 }
10033         }
10034     }
10035 }
10036 \clist_gclear:N \g_@@_cbic_clist
10037 }

```

38 The command \ShowCellNames

```

10038 \NewDocumentCommand \@@_ShowCellNames { }
10039 {
10040     \bool_if:NT \l_@@_in_code_after_bool
10041     {
10042         \pgfpicture
10043         \pgfrememberpicturepositiononpagetrue
10044         \pgf@relevantforpicturesizefalse
10045         \pgfpathrectanglecorners
10046         { \@@_qpoint:n { 1 } }
10047         { \@@_qpoint:n { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } } }
10048         \pgfsetfillopacity { 0.75 }
10049         \pgfsetfillcolor { white }
10050         \pgfusepathqfill
10051         \endpgfpicture
10052     }
10053     \dim_gzero_new:N \g_@@_tmpc_dim
10054     \dim_gzero_new:N \g_@@_tmpd_dim
10055     \dim_gzero_new:N \g_@@_tmpe_dim
10056     \int_step_inline:nn \c@iRow
10057     {
10058         \bool_if:NTF \l_@@_in_code_after_bool
10059         {
10060             \pgfpicture
10061             \pgfrememberpicturepositiononpagetrue
10062             \pgf@relevantforpicturesizefalse
10063             }
10064             { \begin { pgfpicture } }
10065             \@@_qpoint:n { row - ##1 }
10066             \dim_set_eq:NN \l_tmpa_dim \pgf@y
10067             \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
10068             \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
10069             \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
10070             \bool_if:NTF \l_@@_in_code_after_bool
10071             { \endpgfpicture }
10072             { \end { pgfpicture } }
10073             \int_step_inline:nn \c@jCol
10074             {
10075                 \hbox_set:Nn \l_tmpa_box
10076                 {
10077                     \normalfont \Large \sffamily \bfseries
10078                     \bool_if:NTF \l_@@_in_code_after_bool

```



```

10079         { \color { red } }
10080         { \color { red ! 50 } }
10081     ##1 - ####1
10082 }
10083 \bool_if:NTF \l_@@_in_code_after_bool
10084 {
10085     \pgfpicture
10086     \pgfrememberpicturepositiononpagetrue
10087     \pgf@relevantforpicturesizefalse
10088 }
10089 { \begin { pgfpicture } }
10090 \@@_qpoint:n { col - ####1 }
10091 \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
10092 \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
10093 \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
10094 \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
10095 \bool_if:NTF \l_@@_in_code_after_bool
10096 { \endpgfpicture }
10097 { \end { pgfpicture } }
10098 \fp_set:Nn \l_tmpa_fp
10099 {
10100     \fp_min:nn
10101     {
10102         \fp_min:nn
10103         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
10104         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
10105     }
10106     { 1.0 }
10107 }
10108 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
10109 \pgfpicture
10110 \pgfrememberpicturepositiononpagetrue
10111 \pgf@relevantforpicturesizefalse
10112 \pgftransformshift
10113 {
10114     \pgfpoint
10115     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
10116     \g_tmpa_dim
10117 }
10118 \pgfnode
10119 { rectangle }
10120 { center }
10121 { \box_use:N \l_tmpa_box }
10122 { }
10123 { }
10124 \endpgfpicture
10125 }
10126 }
10127 }

```

39 We process the options at package loading

We process the options when the package is loaded (with `\usepackage`) but we recommend to use `\NiceMatrixOptions` instead.

We must process these options after the definition of the environment `{NiceMatrix}` because the option `renew-matrix` executes the code `\cs_set_eq:NN \env@matrix \NiceMatrix`.

Of course, the command `\NiceMatrix` must be defined before such an instruction is executed.

The boolean `\g_@@_footnotehyper_bool` will indicate if the option `footnotehyper` is used.

```

10128 \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.

```

10129 \bool_new:N \g_@@_footnote_bool
10130 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
10131 {
10132   You-have-used-the-key~' \l_keys_key_str '~when~loading~nicematrix~
10133   but~that~key~is~unknown. \\
10134   It~will~be~ignored. \\
10135   For~a~list~of~the~available~keys,~type~H~<return>.
10136 }
10137 {
10138   The~available~keys~are~(in~alphabetic~order):~
10139   footnote,~
10140   footnotehyper,~
10141   messages-for-Overleaf,~
10142   renew-dots~and~
10143   renew-matrix.
10144 }
10145 \keys_define:nn { nicematrix }
10146 {
10147   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
10148   renew-dots .value_forbidden:n = true ,
10149   renew-matrix .code:n = \@@_renew_matrix: ,
10150   renew-matrix .value_forbidden:n = true ,
10151   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
10152   footnote .bool_set:N = \g_@@_footnote_bool ,
10153   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
10154   unknown .code:n = \@@_error:n { Unknown~key~for~package }
10155 }
10156 \ProcessKeyOptions
10157 \@@_msg_new:nn { footnote-with-footnotehyper~package }
10158 {
10159   You~can't~use~the~option~'footnote'~because~the~package~
10160   footnotehyper~has~already~been~loaded.~
10161   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
10162   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10163   of~the~package~footnotehyper.\\
10164   The~package~footnote~won't~be~loaded.
10165 }
10166 \@@_msg_new:nn { footnotehyper-with-footnote~package }
10167 {
10168   You~can't~use~the~option~'footnotehyper'~because~the~package~
10169   footnote~has~already~been~loaded.~
10170   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
10171   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10172   of~the~package~footnote.\\
10173   The~package~footnotehyper~won't~be~loaded.
10174 }
10175 \bool_if:NT \g_@@_footnote_bool
10176 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

10177 \IfClassLoadedTF { beamer }
10178 { \bool_set_false:N \g_@@_footnote_bool }
10179 {
10180   \IfPackageLoadedTF { footnotehyper }
10181   { \@@_error:n { footnote-with-footnotehyper~package } }
10182   { \usepackage { footnote } }

```

```

10183     }
10184   }
10185   \bool_if:NT \g_@@_footnotehyper_bool
10186   {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

10187     \IfClassLoadedTF { beamer }
10188     { \bool_set_false:N \g_@@_footnote_bool }
10189     {
10190       \IfPackageLoadedTF { footnote }
10191       { \@@_error:n { footnotehyper~with~footnote~package } }
10192       { \usepackage { footnotehyper } }
10193     }
10194     \bool_set_true:N \g_@@_footnote_bool
10195   }

```

The flag `\g_@@_footnote_bool` is raised and so, we will only have to test `\g_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

40 About the package underscore

If the user loads the package `underscore`, it must be loaded *before* the package `nicematrix`. If it is loaded after, we raise an error.

```

10196 \bool_new:N \l_@@_underscore_loaded_bool
10197 \IfPackageLoadedT { underscore }
10198 { \bool_set_true:N \l_@@_underscore_loaded_bool }
10199 \AtBeginDocument
10200 {
10201   \bool_if:NF \l_@@_underscore_loaded_bool
10202   {
10203     \IfPackageLoadedT { underscore }
10204     { \@@_error:n { underscore~after~nicematrix } }
10205   }
10206 }

```

41 Compatibility with threeparttable

```

10207 \AtBeginDocument
10208 {
10209   \IfPackageLoadedT { threeparttable }
10210   {
10211     \AddToHook { env / threeparttable / begin }
10212     {
10213       \TPT@hookin { NiceTabular }
10214       \TPT@hookin { NiceTabular* }
10215       \TPT@hookin { NiceTabularX }
10216     }
10217   }
10218 }

```

42 Error messages of the package

When there is a unknown key, maybe the user has tried to use an inexistent “additive syntax” for that key. Of course, in that case, the last character of the name of the key is +.

#1 is a clist of names of sets of keys and #2 is the error message to send.

```

10219 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
10220 {
10221   \str_if_eq:eeTF
10222     { \str_item:Nn \l_keys_key_str { \str_count:N \l_keys_key_str } }
10223     { + }
10224     {
10225       \str_set:Ne \l_tmpa_str
10226         { \str_range:Nnn \l_keys_key_str { 1 } { \str_count:N \l_keys_key_str - 1 } }
10227       \bool_set_false:N \l_tmpa_bool
10228       \clist_map_inline:nn { #1 }
10229         {
10230           \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10231           {
10232             \@@_error:n { key~without~+~exists }
10233             \bool_set_true:N \l_tmpa_bool
10234             \clist_map_break:
10235           }
10236         }
10237       \bool_if:NF \l_tmpa_bool
10238       {
10239         \str_set:Ne \l_keys_key_str { \tl_trim_right_spaces:V \l_tmpa_str }
10240         \@@_unknown_key_i:nn { #1 } { #2 }
10241       }
10242     }
10243     { \@@_unknown_key_i:nn { #1 } { #2 } }
10244 }

```

We try a normalisation of the name of the key, and, when that normal form exists, we add that information in the error message.

The normal form is the lower case form of the key, with all the spaces replaced by hyphens (there is never spaces in the keys of nicematrix).

#1 is a clist of names of sets of keys and #2 is the error message to send.

```

10245 \cs_new_protected:Npn \@@_unknown_key_i:nn #1 #2
10246 {
10247   \str_set_eq:NN \l_tmpa_str \l_keys_key_str
10248   \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
10249   \str_set:Ne \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
10250   \bool_set_false:N \l_tmpa_bool
10251   \clist_map_inline:nn { #1 }
10252     {
10253       \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10254       {
10255         \@@_error:n { key~with~normal~form~exists }
10256         \bool_set_true:N \l_tmpa_bool
10257         \clist_map_break:
10258       }
10259     }
10260   \bool_if:NF \l_tmpa_bool
10261   {
10262     \@@_error:n { #2 }

```

If `messages-for-Overleaf` is not in force, the list of the available keys is not written in the main error message but only in the complement (if the final user presses H). That’s why we write, in all circumstances, the list of the available keys in order to facilitate the work of the systems which analyze the error by IA (such as Prism).

```

10263     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10264     { \msg_info:nn { nicematrix } { #2~+ } }
10265   }
10266 }
10267 \@@_msg_new:nn { key~without~+~exists }
10268 {
10269   The~key~'\tl_trim_right_spaces:V \l_tmpa_str'~exists~but~does~not~accept~an~
10270   additive~syntax~(with~+=).\ \
10271   It~will~be~ignored.\ \
10272 }
10273 \@@_msg_new:nn { key~with~normal~form~exists }
10274 {
10275   No~key~'\l_keys_key_str'.\ \
10276   It~will~be~ignored.\ \
10277   Maybe~you~want~to~use~the~key~'\l_tmpa_str'.
10278 }
10279 \str_const:Ne \c_@@_available_keys_str
10280 {
10281   \bool_if:nT { ! \g_@@_messages_for_Overleaf_bool }
10282   { For~a~list~of~the~available~keys,~type~H~<return>. }
10283 }
10284 \seq_new:N \g_@@_types_of_matrix_seq
10285 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
10286 {
10287   NiceMatrix ,
10288   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
10289 }
10290 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
10291 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_err_too_many_cols:` is triggered. This command raises an error but also tries to give the best information to the user in the error message. The command `\seq_if_in:NoF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

10292 \cs_new_protected:Npn \@@_err_too_many_cols:
10293 {
10294   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
10295   { \@@_fatal:nn { too~many~cols~for~array } }
10296   \int_compare:nNnT \l_@@_last_col_int = { -2 }
10297   { \@@_fatal:n { too~many~cols~for~matrix } }
10298   \int_compare:nNnT \l_@@_last_col_int = { -1 }
10299   { \@@_fatal:n { too~many~cols~for~matrix } }
10300   \bool_if:NF \l_@@_last_col_without_value_bool
10301   { \@@_fatal:n { too~many~cols~for~matrix~with~last~col } }
10302 }

```

The following command must *not* be protected since it's used in an error message.

```

10303 \cs_new:Npn \@@_message_hdotsfor:
10304 {
10305   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
10306   { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
10307     \token_to_str:N \Hbrace \ is~incorrect. }
10308 }
10309 \cs_new_protected:Npn \@@_Hline_in_cell:
10310 { \@@_fatal:n { Misuse~of~Hline } }
10311 \@@_msg_new:nn { Misuse~of~Hline }
10312 {
10313   Misuse~of~Hline. \ \
10314   Error~in~your~row~ \int_eval:N \c@iRow . \ \

```

```

10315 \token_to_str:N \Hline\ (like \token_to_str:N \hline)~must~be~used~only~
10316 at~the~beginning~of~a~row.\\
10317 That~error~is~fatal.
10318 }
10319 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
10320 {
10321   Incompatible~options.\\
10322   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~the~same~time.\\
10323   The~output~will~not~be~reliable.
10324 }
10325 \@@_msg_new:nn { Body-alone }
10326 {
10327   \token_to_str:N \Body\ alone. \\
10328   You~have~used~\token_to_str:N \Body\ without~\token_to_str:N \CodeBefore.\\
10329   That~error~is~fatal.
10330 }
10331 \@@_msg_new:nn { Bad-use-of~CodeBefore }
10332 {
10333   Bad-use-of~\token_to_str:N \CodeBefore. \\
10334   \token_to_str:N \CodeBefore\ must~be~used~only~at~the~beginning~of~
10335   the~environment.\\
10336   That~error~is~fatal.
10337 }
10338 \@@_msg_new:nn { NiceTabularX~probably~required }
10339 {
10340   Incorrect~syntax.\\
10341   You~probably~want~to~use~\{NiceTabularX\}~whose~first~argument~is~the~
10342   disered~width~of~the~tabular.\\
10343   That~error~is~fatal.
10344 }
10345 \@@_msg_new:nn { cellcolor~in~Block }
10346 {
10347   Bad-use-of~\token_to_str:N \cellcolor \\
10348   You~can't~use~\token_to_str:N \cellcolor\ in~\token_to_str:N \Block\
10349   \bool_if:NTF \l_@@_amp_in_blocks_bool
10350     { (but~you~could~use~it~in~a~sub~block~since~'&~in~blocks'~is~in~force) }
10351     { (it's~possible~in~a~sub~block~when~'&~in~blocks'~is~in~force) }
10352     .~Here,~you~should~use~the~key~'fill'~of~the~block.\\
10353     That~command~will~be~ignored.
10354 }
10355 \@@_msg_new:nn { rowcolor~in~Block }
10356 {
10357   Bad-use-of~\token_to_str:N \rowcolor \\
10358   You~can't~use~\token_to_str:N \rowcolor\ in~\token_to_str:N \Block.\\
10359   However,~it's~possible~to~color~the~block~with~its~key~'fill'.\\
10360   That~command~will~be~ignored.
10361 }
10362 \@@_msg_new:nn { key~color~inside }
10363 {
10364   Deleted~key.\\
10365   The~key~'color~inside'~(and~its~alias~'colortbl~like')~has~been~deleted~in
10366   ~'nicematrix'~and~must~not~be~used.\\
10367   This~error~is~fatal.
10368 }
10369 \@@_msg_new:nn { Invalid~argument~for~w }
10370 {
10371   Invalid~argument~for~w.\\
10372   You~have~used~the~type~of~alignment~'#1'~for~your~column~'w'~
10373   but~only~'c',~'r',~'l'~and~'s'~are~allowed~in~a~column~'w'.~
10374   If~you~go~on,~'c'~will~be~used.
10375 }

```

```

10376 \@@_msg_new:nn { invalid-weight }
10377 {
10378   Unknown-key.\
10379   The-key~' \l_keys_key_str '~of~your~column~X~is~unknown~and~will~be~ignored.~
10380   The-available~keys~are:~l,~c,~r,~t~(=p),~m,~b,~V~
10381   \IfPackageLoadedTF { varwidth }
10382     { (since~'varwidth'~is~loaded)~}
10383     { (if~you~load~'varwidth')~}
10384     and~real~numbers~for~the~weight~of~the~X~column.
10385   }
10386 \@@_msg_new:nn { last-col-not-used }
10387 {
10388   Column~not~used.\
10389   The-key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
10390   in~your~\@@_full_name_env: .~
10391   However,~you~can~go~on.
10392 }
10393 \@@_msg_new:nn { too-many-cols-for-matrix-with-last-col }
10394 {
10395   Too-many-columns.\
10396   In~the~row~ \int_eval:n { \c@iRow },~
10397   you~try~to~use~more~columns~
10398   than~allowed~by~your~ \@@_full_name_env: .
10399   \@@_message_hdotsfor: \
10400   The-maximal-number-of~columns~is~ \int_eval:n { \l_@@_last_col_int - 1 }~
10401   (plus~the~exterior~columns).\
10402   But,~maybe,~you~have~forgotten~a~\token_to_str:N \\. \
10403   This~error~is~fatal.
10404 }
10405 \@@_msg_new:nn { too-many-cols-for-matrix }
10406 {
10407   Too-many-columns.\
10408   In~the~row~ \int_eval:n { \c@iRow },~
10409   you~try~to~use~more~columns~than~allowed~by~your~ \@@_full_name_env: .
10410   \@@_message_hdotsfor: \
10411   Recall~that~the~maximal-number-of~columns~for~a~matrix~
10412   (excepted~the~potential~exterior~columns)~is~fixed~by~the~
10413   LaTeX~counter~'MaxMatrixCols'.~
10414   Its~current~value~is~ \int_use:N \c@MaxMatrixCols \
10415   (use~ \token_to_str:N \setcounter \ to~change~that~value).\
10416   But,~maybe,~you~have~forgotten~a~\token_to_str:N \\. \
10417   This~error~is~fatal.
10418 }
10419 \@@_msg_new:nn { too-many-cols-for-array }
10420 {
10421   Too-many-columns.\
10422   In~the~row~ \int_eval:n { \c@iRow },~
10423   ~you~try~to~use~more~columns~than~allowed~by~your~
10424   \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal-number-of~columns~is~
10425   \int_use:N \g_@@_static_num_of_col_int \
10426   \bool_if:nT
10427     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
10428     { (plus~the~exterior~ones)~}
10429   since~the~preamble~is~' \g_@@_user_preamble_tl '.\
10430   But,~maybe,~you~have~forgotten~a~\token_to_str:N \\. \
10431   This~error~is~fatal.
10432 }
10433 \@@_msg_new:nn { columns-not-used }
10434 {
10435   Columns~not~used.\
10436   The~preamble~of~your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '~

```

```

10437     It~announces~ \int_use:N \g_@@_static_num_of_col_int \
10438     columns~but~you~only~used~ \int_use:N \c@jCol .\\
10439     The~columns~you~did~not~used~won't~be~created.\\
10440     You~won't~have~similar~warning~till~the~end~of~the~document.
10441   }
10442 }
10443 \@@_msg_new:nn { Bad~use~of~NiceTabularNotes }
10444 {
10445     Bad~use~of~\token_to_str:N \NiceTabularNotes \\
10446     \token_to_str:N~\NiceTabularNotes\ should~be~used~only~
10447     after~at~tabular~which~uses~`notes/no-print`. \\
10448     That~command~will~be~ignored.
10449 }
10450 \@@_msg_new:nn { empty~preamble }
10451 {
10452     Empty~preamble.\\
10453     The~preamble~of~your~ \@@_full_name_env: \ is~empty.\\
10454     This~error~is~fatal.
10455 }
10456 \@@_msg_new:nn { in~first~col }
10457 {
10458     Erroneous~use.\\
10459     You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
10460     That~command~will~be~ignored.\\
10461     You~can~try~to~delete~the~key~'first-col'.
10462 }
10463 \@@_msg_new:nn { in~last~col }
10464 {
10465     Erroneous~use.\\
10466     You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
10467     That~command~will~be~ignored.\\
10468     You~can~try~to~delete~the~key~'last-col'.
10469 }
10470 \@@_msg_new:nn { in~first~row }
10471 {
10472     Erroneous~use.\\
10473     You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
10474     That~command~will~be~ignored.\\
10475     You~can~try~to~delete~the~key~'first-row'.
10476 }
10477 \@@_msg_new:nn { in~last~row }
10478 {
10479     Erroneous~use.\\
10480     You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
10481     That~command~will~be~ignored.\\
10482     You~can~try~to~delete~the~key~'last-row'.
10483 }
10484 \@@_msg_new:nn { TopRule~without~booktabs }
10485 {
10486     Erroneous~use.\\
10487     You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
10488     You~should~load~'booktabs'~(before~or~after~'nicematrix').\\
10489     That~command~will~be~ignored.
10490 }
10491 \@@_msg_new:nn{ rotate~in~p~col }
10492 {
10493     \token_to_str:N \rotate\ forbidden.\\
10494     You~should~not~use~\token_to_str:N \rotate\ in~a~column~of~type~'p',~
10495     'b',~'m'\IfPackageLoadedTF { varwidth } { ,~'X'~or~'V' } { ~or~'X'}.~
10496     If~you~go~on,~maybe~you~won't~have~the~expected~output.
10497 }

```



```

10498 \@@_msg_new:nn { TopRule~without~tikz }
10499 {
10500   Erroneous~use.\\
10501   You~can't~use~the~command~ #1 because~TikZ~is~not~loaded.\\
10502   You~should~load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.\\
10503   \IfPackageLoadedF { booktabs }
10504   { You~should~also~load~'booktabs'.\\ }
10505   That~command~will~be~ignored.
10506 }
10507 \@@_msg_new:nn { caption~outside~float }
10508 {
10509   Key~caption~forbidden.\\
10510   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
10511   environment~(such~as~\{table\}).~This~key~will~be~ignored.
10512 }
10513 \@@_msg_new:nn { short~caption~without~caption }
10514 {
10515   You~should~not~use~the~key~'short~caption'~without~'caption'.~
10516   However,~your~'short~caption'~will~be~used~as~'caption'.
10517 }
10518 \@@_msg_new:nn { double~closing~delimiter }
10519 {
10520   Double~delimiter.\\
10521   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
10522   delimiter.~This~delimiter~will~be~ignored.\\
10523   You~can~try~to~use~\token_to_str:N \SubMatrix\ in~the~\token_to_str:N \CodeAfter.
10524 }
10525 \@@_msg_new:nn { delimiter~after~opening }
10526 {
10527   Double~delimiter.\\
10528   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
10529   delimiter.~That~delimiter~will~be~ignored.
10530 }
10531 \@@_msg_new:nn { bad~option~for~line~style }
10532 {
10533   Bad~line~style.\\
10534   Since~you~haven't~loaded~TikZ,~the~only~value~you~can~give~to~'line~style'~
10535   is~'standard'.~That~key~will~be~ignored.\\
10536   You~can~load~TikZ~with~\token_to_str:N \usepackage \{tikz\},~
10537   before~or~after~'nicematrix'.\\
10538 }
10539 \@@_msg_new:nn { corners~with~no~cell~nodes }
10540 {
10541   Incompatible~keys.\\
10542   You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
10543   is~in~force~(you~should~deactive~the~key~'no~cell~nodes'~whose~only~goal~
10544   is~to~speed~up~compilation).\\
10545   If~you~go~on,~that~key~will~be~ignored.
10546 }
10547 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
10548 {
10549   Incompatible~keys.\\
10550   You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
10551   is~in~force~(you~should~deactive~the~key~'no~cell~nodes'~whose~only~goal~
10552   is~to~speed~up~compilation).\\
10553   If~you~go~on,~those~extra~nodes~won't~be~created.
10554 }
10555 \@@_msg_new:nn { Identical~notes~in~caption }
10556 {
10557   Identical~tabular~notes.\\

```

```

10558     You~can't~put~several~notes~with~the~same~content~in~
10559     \token_to_str:N \caption \ (but~it's~possible~in~the~main~tabular).\
10560     If~you~go~on,~the~output~will~probably~be~erroneous.
10561 }

10562 \@@_msg_new:nn { tabularnote~below~the~tabular }
10563 {
10564     \token_to_str:N \tabularnote \ forbidden\
10565     You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
10566     of~your~tabular~because~the~caption~will~be~composed~below~
10567     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
10568     key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .\
10569     Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
10570     no~similar~error~will~raised~in~this~document.
10571 }

10572 \@@_msg_new:nn { Unknown~key~for~rules }
10573 {
10574     Unknown~key.\
10575     There~are~only~three~keys~available~here:~'width',~'color'~and~
10576     'fix~vertex'.\
10577     Your~key~' \l_keys_key_str '~will~be~ignored.
10578 }

10579 \@@_msg_new:nn { Unknown~key~for~trees }
10580 {
10581     Unknown~key.\
10582     There~are~only~three~keys~available~here:~width~color~and~
10583     rounded~corners.\
10584     Your~key~' \l_keys_key_str '~will~be~ignored.
10585 }
10586 % \end{macrocode}
10587 %
10588 %
10589 % \begin{macrocode}
10590 \@@_msg_new:nn { Unknown~key~for~Hbrace }
10591 {
10592     Unknown~key.\
10593     You~have~used~the~key~' \l_keys_key_str '~but~the~only~
10594     keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
10595     and~ \token_to_str:N \Vbrace \ are:~'brace~shift(+)',~'color',~
10596     'horizontal~label(s)',~'shorten'~'shorten~end'~
10597     and~'shorten~start'.\
10598     That~error~is~fatal.
10599 }

10600 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
10601 {
10602     Unknown~key.\
10603     There~is~only~two~keys~available~here:~
10604     'empty'~and~'not~empty'.\
10605     Your~key~' \l_keys_key_str '~will~be~ignored.
10606 }

10607 \@@_msg_new:nn { Unknown~key~for~rotate }
10608 {
10609     Unknown~key.\
10610     The~only~key~available~here~is~'c'.\
10611     Your~key~' \l_keys_key_str '~will~be~ignored.
10612 }

10613 \@@_msg_new:nnn { Unknown~key~for~custom~line }
10614 {
10615     Unknown~key.\
10616     The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'~.~
10617     It~you~go~on,~you~will~probably~have~other~errors. \
10618     \c_@@_available_keys_str

```

```

10619 }
10620 {
10621   The-available-keys-are-(in~alphabetic~order):~
10622   ccommand,~
10623   color,~
10624   command,~
10625   dotted,~
10626   letter,~
10627   multiplicity,~
10628   sep-color,~
10629   tikz,~and~total-width.
10630 }
10631 \@@_msg_new:nnn { Unknown~key~for~default~line }
10632 {
10633   Unknown~key.\\
10634   The~key~' \l_keys_key_str '~is~unknown~in~a~'default~line'.~
10635   It~you~go~on,~you~will~probably~have~other~errors. \\
10636   \c_@@_available_keys_str
10637 }
10638 {
10639   The-available-keys-are-(in~alphabetic~order):~
10640   color,~
10641   dotted,~
10642   multiplicity,~
10643   sep-color,~
10644   tikz,~and~total-width.
10645 }
10646 \@@_msg_new:nnn { Unknown~key~for~xdots }
10647 {
10648   Unknown~key.\\
10649   The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
10650   \c_@@_available_keys_str
10651 }
10652 {
10653   The-available-keys-are-(in~alphabetic~order):~
10654   'color',~
10655   'horizontal(s)-labels',~
10656   'inter',~
10657   'line-style',~
10658   'nullify',~
10659   'radius',~
10660   'shorten',~
10661   'shorten-end'~and~'shorten-start'.
10662 }
10663 \@@_msg_new:nn { Unknown~key~for~rowcolors }
10664 {
10665   Unknown~key.\\
10666   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect~blocks'~
10667   (and~you~try~to~use~' \l_keys_key_str '~)\\
10668   That~key~will~be~ignored.
10669 }
10670 \@@_msg_new:nn { Col~outside~tabular~in~trees }
10671 {
10672   Error~with~'draw~trees~in~col' \\
10673   The~number~of~column~'#1'~is~outside~your~tabular~since~the~last~column~
10674   is~\int_use:N \c@jCol. \\
10675   It~will~be~ignored.
10676 }
10677 \@@_msg_new:nn { Last~col~in~trees }
10678 {
10679   Error~with~'draw~trees~in~col' \\
10680   You~can't~use~'draw~trees~in~col'~with~the~column~'#1'~

```

```

10681     because~it's~the~last~column~of~your~tabular. \\
10682     It~will~be~ignored.
10683 }
10684 \@@_msg_new:nn { label~without~caption }
10685 {
10686     You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
10687     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
10688 }
10689 \@@_msg_new:nn { W~warning }
10690 {
10691     Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
10692     (row~ \int_use:N \c@iRow ).
10693 }
10694 \@@_msg_new:nn { Construct~too~large }
10695 {
10696     Construct~too~large.\\
10697     Your~command~ \token_to_str:N #1
10698     can't~be~drawn~because~your~matrix~is~too~small.\\
10699     That~command~will~be~ignored.
10700 }
10701 \@@_msg_new:nn { underscore~after~nicematrix }
10702 {
10703     Problem~with~'underscore'.\\
10704     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
10705     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
10706     ' \token_to_str:N \Cdots \token_to_str:N _
10707     \{ n \token_to_str:N \text \{ ~times \} \}' .
10708 }
10709 \@@_msg_new:nn { ampersand~in~light-syntax }
10710 {
10711     Ampersand~forbidden.\\
10712     You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
10713     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
10714 }
10715 \@@_msg_new:nn { double~backslash~in~light-syntax }
10716 {
10717     Double~backslash~forbidden.\\
10718     You~can't~use~ \token_to_str:N \\
10719     ~to~separate~rows~because~the~key~'light-syntax'~
10720     is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
10721     (set~by~the~key~'end-of-row').~This~error~is~fatal.
10722 }
10723 \@@_msg_new:nn { hlines~with~color }
10724 {
10725     Incompatible~keys.\\
10726     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
10727     \token_to_str:N \Block \ when~the~key~'color'~or~'draw'~is~used.\\
10728     However,~you~can~put~several~commands~ \token_to_str:N \Block.\\
10729     Your~key~will~be~discarded.
10730 }
10731 \@@_msg_new:nn { bad~value~for~baseline }
10732 {
10733     Bad~value~for~baseline.\\
10734     The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
10735     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
10736     \int_use:N \g_@@_row_total_int \ or~equal~to~'t',~'c'~or~'b'~or~of~
10737     the~form~'line-i'.\\
10738     A~value~of~1~will~be~used.
10739 }
10740 \@@_msg_new:nn { bad~value~for~baseline~line }

```

```

10741 {
10742   Bad-value-for-baseline-with-line.\
10743   The-value-given-to-'baseline'~( \int_use:N \l_tmpa_int )~is-not~
10744   valid.~The-number-of-the-line-must-be-between~1~and~
10745   \int_eval:n { \c@iRow + 1 } \
10746   A-value-of-'line-1'~will-be-used.
10747 }
10748 \@@_msg_new:nn { detection-of-empty-cells }
10749 {
10750   Problem-with-'not-empty'\
10751   For-technical-reasons,~you-must-activate~
10752   'create-cell-nodes'~in~ \token_to_str:N \CodeBefore \
10753   in-order-to-use-the-key~' \l_keys_key_str '.\
10754   That-key-will-be-ignored.
10755 }
10756 \@@_msg_new:nn { siunitx-not-loaded }
10757 {
10758   siunitx-not-loaded\
10759   You-can't-use-the-columns~'S'~because~'siunitx'~is-not-loaded.\
10760   That-error-is-fatal.\
10761   You-can-load~'siunitx'~with~\token_to_str:N \usepackage \{siunitx\},~
10762   before-or-after~'nicematrix'. \
10763 }
10764 \@@_msg_new:nn { Invalid-name }
10765 {
10766   Invalid-name.\
10767   You-can't-give-the-name~' \l_keys_value_tl '~to-a~ \token_to_str:N
10768   \SubMatrix \ of-your~ \@@_full_name_env: .\
10769   A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\
10770   This-key-will-be-ignored.
10771 }
10772 \@@_msg_new:nn { Hbrace-not-allowed }
10773 {
10774   Command-not-allowed.\
10775   You-can't-use-the-command~ \token_to_str:N #1
10776   because-you-have-not-loaded~
10777   \IfPackageLoadedTF { tikz }
10778   { the-TikZ-library~'decorations.pathreplacing'~Use~ }
10779   { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
10780   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \
10781   That-command-will-be-ignored.
10782 }
10783 \@@_msg_new:nn { Vbrace-not-allowed }
10784 {
10785   Command-not-allowed.\
10786   You-can't-use-the-command~ \token_to_str:N \Vbrace \
10787   because-you-have-not-loaded-TikZ~
10788   and-the-TikZ-library~'decorations.pathreplacing'.\
10789   Use: ~\token_to_str:N \usepackage \{tikz\}~
10790   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \
10791   That-command-will-be-ignored.
10792 }
10793 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
10794 {
10795   Wrong-line.\
10796   You-try-to-draw-a~#1~line-of-number~'#2'~in-a~
10797   \token_to_str:N \SubMatrix \ of-your~ \@@_full_name_env: \ but-that~
10798   number-is-not-valid.~It-will-be-ignored.
10799 }
10800 \@@_msg_new:nn { Impossible-delimiter }
10801 {

```

```

10802     Impossible~delimiter.\\
10803     It's~impossible~to~draw~the~#1~delimiter~of~your~
10804     \token_to_str:N \SubMatrix \ because~all~the~cells~are~empty~
10805     in~that~column.
10806     \bool_if:NT \l_@@_submatrix_slim_bool
10807     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
10808     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
10809 }
10810 \@@_msg_new:nnn { width~without~X~columns }
10811 {
10812     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
10813     the~preamble~(' \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .\\
10814     That~key~will~be~ignored.
10815 }
10816 {
10817     This~message~is~the~message~'width~without~X~columns'~
10818     of~the~module~'nicematrix'.~
10819     The~experimented~users~can~disable~that~message~with~
10820     \token_to_str:N \msg_redirect_name:nnn .\\
10821 }
10822
10823 \@@_msg_new:nn { key~multiplicity~with~dotted }
10824 {
10825     Incompatible~keys. \\
10826     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
10827     in~a~'custom-line'~.~They~are~incompatible. \\
10828     The~key~'multiplicity'~will~be~discarded.
10829 }
10830 \@@_msg_new:nn { empty~environment }
10831 {
10832     Empty~environment.\\
10833     Your~ \@@_full_name_env: \ is~empty.~This~error~is~fatal.
10834 }
10835 \@@_msg_new:nn { No~letter~and~no~command }
10836 {
10837     Erroneous~use.\\
10838     Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
10839     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10840     '~'command'~(to~draw~horizontal~rules).\\
10841     However,~you~can~go~on.
10842 }
10843 \@@_msg_new:nn { Forbidden~letter }
10844 {
10845     Forbidden~letter.\\
10846     You~can't~use~the~letter~'#1'~for~a~customized~line.~
10847     It~will~be~ignored.\\
10848     The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10849 }
10850 \@@_msg_new:nn { Several~letters }
10851 {
10852     Wrong~name.\\
10853     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10854     have~used~'\l_@@_letter_str ').\\
10855     It~will~be~ignored.
10856 }
10857 \@@_msg_new:nn { Delimiter~with~small }
10858 {
10859     Delimiter~forbidden.\\
10860     You~can't~put~a~delimiter~in~the~preamble~of~your~
10861     \@@_full_name_env: \
10862     because~the~key~'small'~is~in~force.\\

```

```

10863     This-error-is-fatal.
10864 }
10865 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
10866 {
10867     Unknown-cell.\
10868     Your-command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10869     the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10870     can't-be-executed-because-a-cell-doesn't-exist.\
10871     This-command~ \token_to_str:N \line \ will-be-ignored.
10872 }
10873 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
10874 {
10875     Duplicate-name.\
10876     The-name- '#1' ~is-already-used-for-a~ \token_to_str:N \SubMatrix \
10877     in-this~ \@@_full_name_env: .\
10878     This-key-will-be-ignored.\
10879     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10880     { For-a-list-of-the-names-already-used,~type-H~<return>. }
10881 }
10882 {
10883     The-names-already-defined-in-this~ \@@_full_name_env: \ are:~
10884     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10885 }
10886 \@@_msg_new:nn { r-or-l-with-preamble }
10887 {
10888     Erroneous-use.\
10889     You-can't-use-the-key~ ' \l_keys_key_str '~in-your~ \@@_full_name_env: .~
10890     You-must-specify-the-alignment-of-your-columns-with-the-preamble-of~
10891     your~ \@@_full_name_env: .\
10892     This-key-will-be-ignored.
10893 }
10894 \@@_msg_new:nn { Hdotsfor-in-col-0 }
10895 {
10896     Erroneous-use.\
10897     You-can't-use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\
10898     in-an-exterior-column-of~
10899     the-array.~This-error-is-fatal.
10900 }
10901 \@@_msg_new:nn { bad-corner }
10902 {
10903     Bad-corner.\
10904     #1-is-an-incorrect-specification-for-a-corner~(in-the-key~
10905     'corners').~The-available-values-are:~NW,~SW,~NE-and~SE.\
10906     This-specification-of-corner-will-be-ignored.
10907 }
10908 \@@_msg_new:nn { bad-border }
10909 {
10910     Bad-border.\
10911     \l_keys_key_str \space ~is-an-incorrect-specification-for-a-border~
10912     (in-the-key~'borders'~of-the-command~ \token_to_str:N \Block ).~
10913     The-available-values-are:~left,~right,~top-and-bottom~(and-you-can~
10914     also-use-the-key~'tikz'
10915     \IfPackageLoadedF { tikz }
10916     { ~if-you-load-the-LaTeX-package~'tikz' } ).\
10917     This-specification-of-border-will-be-ignored.
10918 }
10919 \@@_msg_new:nn { TikzEveryCell-without-tikz }
10920 {
10921     TikZ-not-loaded.\
10922     You-can't-use~ \token_to_str:N \TikzEveryCell \
10923     because-you-have-not-loaded-tikz.\

```

```

10924     You~can~load~'tikz'~with~\token_to_str:N \usepackage \{tikz\},~
10925     before~or~after~'nicematrix'. \\
10926     This~command~will~be~ignored.
10927 }
10928 \@@_msg_new:nn { tikz~key~without~tikz }
10929 {
10930     TikZ~not~loaded. \\
10931     You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N
10932     \Block '~because~you~have~not~loaded~tikz. \\
10933     You~can~load~'tikz'~with~\token_to_str:N \usepackage \{tikz\},~
10934     before~or~after~'nicematrix'. \\
10935     This~key~will~be~ignored.
10936 }
10937 \@@_msg_new:nn { Bad~argument~for~Block }
10938 {
10939     Bad~argument. \\
10940     The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
10941     be~of~the~form~'i-j'~(or~totally~empty)~and~you~have~used:~
10942     '#1'. \\
10943     If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono~cell~(as~if~
10944     the~argument~was~empty).
10945 }
10946 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
10947 {
10948     Erroneous~use. \\
10949     In~the~ \@@_full_name_env: ,~you~must~use~the~key~
10950     'last~col'~without~value. \\
10951     However,~you~can~go~on~for~this~time~
10952     (the~value~' \l_keys_value_tl '~will~be~ignored).
10953 }
10954 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
10955 {
10956     Erroneous~use. \\
10957     In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
10958     'last~col'~without~value. \\
10959     However,~you~can~go~on~for~this~time~
10960     (the~value~' \l_keys_value_tl '~will~be~ignored).
10961 }
10962 \@@_msg_new:nn { Block~too~large~1 }
10963 {
10964     Block~too~large. \\
10965     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
10966     too~small~for~that~block. \\
10967     This~block~and~maybe~others~will~be~ignored.
10968 }
10969 \@@_msg_new:nn { Block~too~large~2 }
10970 {
10971     Block~too~large. \\
10972     The~preamble~of~your~ \@@_full_name_env: \ announces~ \int_use:N
10973     \g_@@_static_num_of_col_int \
10974     columns~but~you~use~only~ \int_use:N \c_jCol \ and~that's~why~a~block~
10975     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
10976     (&)~at~the~end~of~the~first~row~of~your~ \@@_full_name_env: . \\
10977     This~block~and~maybe~others~will~be~ignored.
10978 }
10979 \@@_msg_new:nn { unknown~column~type }
10980 {
10981     Bad~column~type. \\
10982     The~column~type~'#1'~in~your~ \@@_full_name_env: \
10983     is~unknown. \\
10984     This~error~is~fatal.

```



```

10985 }
10986 \@@_msg_new:nn { unknown~column-type-multicolumn }
10987 {
10988   Bad~column~type. \\\
10989   The~column~type~'#1'~in~the~command~\token_to_str:N \multicolumn \
10990   ~of~your~ \@@_full_name_env: \
10991   is~unknown. \\\
10992   This~error~is~fatal.
10993 }
10994 \@@_msg_new:nn { unknown~column-type-S }
10995 {
10996   Bad~column~type. \\\
10997   The~column~type~'S'~in~your~ \@@_full_name_env: \ is~unknown. \\\
10998   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10999   load~that~package. \\\
11000   This~error~is~fatal.
11001 }
11002 \@@_msg_new:nn { unknown~column-type-S-multicolumn }
11003 {
11004   Bad~column~type. \\\
11005   The~column~type~'S'~in~the~command~\token_to_str:N \multicolumn \
11006   of~your~ \@@_full_name_env: \ is~unknown. \\\
11007   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
11008   load~that~package. \\\
11009   This~error~is~fatal.
11010 }
11011 \@@_msg_new:nn { tabularnote~forbidden }
11012 {
11013   Forbidden~command. \\\
11014   You~can't~use~the~command~ \token_to_str:N \tabularnote \
11015   ~here.~This~command~is~available~only~in~
11016   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
11017   the~argument~of~a~command~\token_to_str:N \caption \ included~
11018   in~an~environment~\{table\}. \\\
11019   This~command~will~be~ignored.
11020 }
11021 \@@_msg_new:nn { borders~forbidden }
11022 {
11023   Forbidden~key.\\
11024   You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \
11025   because~the~option~'rounded~corners'~
11026   is~in~force~with~a~non~zero~value.\\
11027   This~key~will~be~ignored.
11028 }
11029 \@@_msg_new:nn { bottomrule~without~booktabs }
11030 {
11031   booktabs~not~loaded.\\
11032   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
11033   loaded~'booktabs'.~You~should~load~'booktabs',~before~or~
11034   after~'nicematrix'.\\
11035   This~key~will~be~ignored.
11036 }
11037 \@@_msg_new:nn { enumitem~not~loaded }
11038 {
11039   enumitem~not~loaded. \\\
11040   You~can't~use~the~command~ \token_to_str:N \tabularnote \
11041   ~because~you~haven't~loaded~'enumitem'.~We~should~load~it~
11042   (before~or~after~'nicematrix').\\
11043   All~the~commands~ \token_to_str:N \tabularnote \ will~be~
11044   ignored~in~the~document.
11045 }

```

```

11046 \@@_msg_new:nn { tikz-without-tikz }
11047 {
11048   TikZ~not~loaded. \\
11049   You~can't~use~the~key~'tikz'~here~because~TikZ~is~not~
11050   loaded.~If~you~go~on,~that~key~will~be~ignored.
11051 }
11052 \@@_msg_new:nn { tikz-in-custom-line-without-tikz }
11053 {
11054   TikZ~not~loaded. \\
11055   You~have~used~the~key~'tikz'~in~the~definition~of~a~
11056   customized~line~(with~'custom-line')~but~TikZ~is~not~loaded.~
11057   You~can~go~on~but~you~will~have~another~error~if~you~actually~
11058   use~that~custom~line.
11059 }
11060 \@@_msg_new:nn { tikz-in-borders-without-tikz }
11061 {
11062   TikZ~not~loaded. \\
11063   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
11064   command~' \token_to_str:N \Block ')~but~TikZ~is~not~loaded.~
11065   That~key~will~be~ignored.
11066 }
11067 \@@_msg_new:nn { color-in-custom-line-with-tikz }
11068 {
11069   Erroneous~use.\\
11070   In~a~'custom-line',~you~have~used~both~'tikz'~and~'color',~
11071   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
11072   The~key~'color'~will~be~discarded.
11073 }
11074 \@@_msg_new:nn { Wrong-last-row }
11075 {
11076   Wrong~number.\\
11077   You~have~used~'last-row= \int_use:N \l_@@_last_row_int '~but~your~
11078   \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
11079   If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
11080   last~row~but~you~should~correct~your~code.~You~can~avoid~this~
11081   problem~by~using~'last-row'~without~value~(more~compilations~
11082   might~be~necessary).
11083 }
11084 \@@_msg_new:nn { Yet-in-env }
11085 {
11086   Nested~environments.\\
11087   Environments~of~nicematrix~can't~be~nested.~However~you~
11088   can~insert,~for~example,~a~\{tabular\}~in~a~\{NiceTabular\}~
11089   or~a~\{NiceTabular\}~in~a~\{tabular\}.~You~can~also~compose~
11090   an~environment~of~nicematrix~in~a~box~of~LaTeX~and~insert~
11091   that~box~in~another~environment~of~nicematrix.\\
11092   This~error~is~fatal.
11093 }
11094 \@@_msg_new:nn { Outside-math-mode }
11095 {
11096   Outside~math~mode.\\
11097   The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
11098   (and~not~in~ \token_to_str:N \vcenter ).\\
11099   This~error~is~fatal.
11100 }
11101 \@@_msg_new:nn { One-letter~allowed }
11102 {
11103   Bad~name.\\
11104   The~value~of~key~' \l_keys_key_str '~must~be~of~length~1~and~
11105   you~have~used~' \l_keys_value_tl '~.\\
11106   It~will~be~ignored.

```

```

11107 }
11108 \@@_msg_new:nn { TabularNote~in~CodeAfter }
11109 {
11110   Environment~\{TabularNote\}~forbidden.\\
11111   You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
11112   but~*before*~the~ \token_to_str:N \CodeAfter . \\
11113   This~environment~\{TabularNote\}~will~be~ignored.
11114 }
11115 \@@_msg_new:nn { varwidth~not~loaded }
11116 {
11117   varwidth~not~loaded.\\
11118   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
11119   loaded.~You~should~load~'varwidth',~before~or~after~'nicematrix'. \\
11120   Your~column~will~behave~like~'p'.
11121 }
11122 \@@_msg_new:nn { varwidth~not~loaded~in~X }
11123 {
11124   varwidth~not~loaded.\\
11125   You~can't~use~the~key~'V'~in~your~column~'X'~
11126   because~'varwidth'~is~not~loaded.~You~should~load~'varwidth',~
11127   before~or~after~'nicematrix'.\\
11128   It~will~be~ignored. \\
11129 }
11130 \@@_msg_new:nnn { Unknown~key~for~a~rule }
11131 {
11132   Unknown~key.\\
11133   Your~key~' \l_keys_key_str '~is~unknown~for~a~rule.\\
11134   \c_@@_available_keys_str
11135 }
11136 {
11137   The~available~keys~are:~color,~multiplicity,~sep~color,~tikz~
11138   and~total~width~(meaningful~only~in~cunjonction~with~'tikz').
11139 }

```

If fact, there is also the key dotted but it won't be very useful since we provide \hdottedline, \cdottedline and the letter :.

```

11140 \@@_msg_new:nnn { Unknown~key~for~Block }
11141 {
11142   Unknown~key. \\
11143   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11144   \token_to_str:N \Block . \\
11145   It~will~be~ignored. \\
11146   \c_@@_available_keys_str
11147 }
11148 {
11149   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
11150   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~name,~
11151   opacity,~rounded~corners,~r,~respect~arraystretch,~rules/width,~t,~T,~tikz,~
11152   transparent~and~vlines.
11153 }
11154 \@@_msg_new:nnn { Unknown~key~for~Brace }
11155 {
11156   Unknown~key.\\
11157   The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
11158   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
11159   It~will~be~ignored. \\
11160   \c_@@_available_keys_str
11161 }
11162 {
11163   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
11164   right~shorten,~shorten~(which~fixes~both~left~shorten~and~

```

```

11165     right-shorten)~and-yshift.
11166 }
11167 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
11168 {
11169     Unknown~key.\\
11170     The~key~' \l_keys_key_str '~is~unknown.\\
11171     It~will~be~ignored. \\
11172     \c_@@_available_keys_str
11173 }
11174 {
11175     The~available~keys~are~(in~alphabetic~order):~
11176     delimiters/color,~
11177     rules~(with~the~subkeys~'color'~and~'width'),~
11178     sub-matrix~(several~subkeys)~
11179     and~xdots~(several~subkeys).~
11180     The~latter~is~for~the~command~ \token_to_str:N \line .
11181 }
11182 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
11183 {
11184     Unknown~key.\\
11185     The~key~' \l_keys_key_str '~is~unknown.\\
11186     It~will~be~ignored. \\
11187     \c_@@_available_keys_str
11188 }
11189 {
11190     The~available~keys~are~(in~alphabetic~order):~
11191     create-cell-nodes,~
11192     delimiters/color~and~
11193     sub-matrix~(several~subkeys).
11194 }
11195 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
11196 {
11197     Unknown~key.\\
11198     The~key~' \l_keys_key_str '~is~unknown.\\
11199     That~key~will~be~ignored. \\
11200     \c_@@_available_keys_str
11201 }
11202 {
11203     The~available~keys~are~(in~alphabetic~order):~
11204     'delimiters/color',~
11205     'extra-height',~
11206     'hlines',~
11207     'hvlines',~
11208     'left-xshift',~
11209     'name',~
11210     'right-xshift',~
11211     'rules'~(with~the~subkeys~'color'~and~'width'),~
11212     'slim',~
11213     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
11214     and~'right-xshift').\\
11215 }
11216 \@@_msg_new:nnn { Unknown~key~for~notes }
11217 {
11218     Unknown~key.\\
11219     The~key~' \l_keys_key_str '~is~unknown.\\
11220     That~key~will~be~ignored. \\
11221     \c_@@_available_keys_str
11222 }
11223 {
11224     The~available~keys~are~(in~alphabetic~order):~
11225     bottomrule,~
11226     code-after,~

```

```

11227     code-before(+),~
11228     detect-duplicates,~
11229     enumitem-keys,~
11230     enumitem-keys-para,~
11231     para,~
11232     label-in-list,~
11233     label-in-tabular~and~
11234     style.
11235 }

11236 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
11237 {
11238     Unknown~key.\\
11239     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11240     \token_to_str:N \RowStyle . \\
11241     That~key~will~be~ignored. \\
11242     \c_@@_available_keys_str
11243 }
11244 {
11245     The~available~keys~are~(in~alphabetic~order):~
11246     bold,~
11247     cell-space-top-limit(+),~
11248     cell-space-bottom-limit(+),~
11249     cell-space-limits(+),~
11250     color,~
11251     fill~(alias:~rowcolor),~
11252     nb-rows,~
11253     opacity~and~
11254     rounded-corners.
11255 }

11256 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
11257 {
11258     Unknown~key.\\
11259     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11260     \token_to_str:N \NiceMatrixOptions . \\
11261     That~key~will~be~ignored. \\
11262     \c_@@_available_keys_str
11263 }
11264 {
11265     The~available~keys~are~(in~alphabetic~order):~
11266     &~in~blocks,~
11267     allow-duplicate-names,~
11268     ampersand-in-blocks,~
11269     caption-above,~
11270     cell-space-bottom-limit(+),~
11271     cell-space-limits(+),~
11272     cell-space-top-limit(+),~
11273     code-for-first-col(+),~
11274     code-for-first-row(+),~
11275     code-for-last-col(+),~
11276     code-for-last-row(+),~
11277     corners,~
11278     custom-key,~
11279     create-extra-nodes,~
11280     create-medium-nodes,~
11281     create-large-nodes,~
11282     custom-line,~
11283     delimiters~(several~subkeys),~
11284     end-of-row,~
11285     first-col,~
11286     first-row,~
11287     hlines,~
11288     hvlines,~
11289     hvlines-except-borders,~

```

```

11290     last-col,~
11291     last-row,~
11292     left-margin,~
11293     light-syntax,~
11294     light-syntax-expanded,~
11295     matrix/columns-type,~
11296     no-cell-nodes,~
11297     notes~(several~subkeys),~
11298     nullify-dots,~
11299     pgf-node-code,~
11300     renew-dots,~
11301     renew-matrix,~
11302     respect-arraystretch,~
11303     rounded-corners,~
11304     right-margin,~
11305     rules~(with~the~subkeys~'color'~and~'width'),~
11306     small,~
11307     sub-matrix~(several~subkeys),~
11308     vlines,~
11309     xdots~(several~subkeys).
11310 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no `l` and `r`.

```

11311 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
11312 {
11313   Unknown~key.\\
11314   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11315   \{NiceArray\}. \\
11316   That~key~will~be~ignored. \\
11317   \c_@@_available_keys_str
11318 }
11319 {
11320   The~available~keys~are~(in~alphabetic~order):~
11321   &~in~blocks,~
11322   ampersand~in~blocks,~
11323   b,~
11324   baseline,~
11325   c,~
11326   cell-space-bottom-limit,~
11327   cell-space-limits,~
11328   cell-space-top-limit,~
11329   code~after,~
11330   code~for~first~col(+),~
11331   code~for~first~row(+),~
11332   code~for~last~col(+),~
11333   code~for~last~row(+),~
11334   columns~width,~
11335   corners,~
11336   create~blocks~in~col,~
11337   create~extra~nodes,~
11338   create~medium~nodes,~
11339   create~large~nodes,~
11340   extra~left~margin,~
11341   extra~right~margin,~
11342   first~col,~
11343   first~row,~
11344   hlines,~
11345   hvlines,~
11346   hvlines~except~borders,~
11347   last~col,~
11348   last~row,~
11349   left~margin,~
11350   light-syntax,~

```

```

11351 light-syntax-expanded,~
11352 name,~
11353 no-cell-nodes,~
11354 nullify-dots,~
11355 pgf-node-code,~
11356 renew-dots,~
11357 respect-arraystretch,~
11358 right-margin,~
11359 rounded-corners,~
11360 rules~(with~the~subkeys~'color'~and~'width'),~
11361 small,~
11362 t,~
11363 vl原因,~
11364 xdots/color,~
11365 xdots/shorten-start(+),~
11366 xdots/shorten-end(+),~
11367 xdots/shorten(+)-and~
11368 xdots/line-style.
11369 }

```

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`).

```

11370 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
11371 {
11372   Unknown~key.\\
11373   The~key~' \l_keys_key_str '~is~unknown~for~the~
11374   \@@_full_name_env: . \\
11375   That~key~will~be~ignored. \\
11376   \c_@@_available_keys_str
11377 }
11378 {
11379   The~available~keys~are~(in~alphabetic~order):~
11380   &~in~blocks,~
11381   ampersand~in~blocks,~
11382   b,~
11383   baseline,~
11384   c,~
11385   cell-space-bottom-limit,~
11386   cell-space-limits,~
11387   cell-space-top-limit,~
11388   code-after,~
11389   code-for-first-col(+),~
11390   code-for-first-row(+),~
11391   code-for-last-col(+),~
11392   code-for-last-row(+),~
11393   columns-type,~
11394   columns-width,~
11395   corners,~
11396   create-blocks-in-col,~
11397   create-extra-nodes,~
11398   create-medium-nodes,~
11399   create-large-nodes,~
11400   extra-left-margin,~
11401   extra-right-margin,~
11402   first-col,~
11403   first-row,~
11404   hlines,~
11405   hvlines,~
11406   hvlines-except-borders,~
11407   l,~
11408   last-col,~
11409   last-row,~
11410   left-margin,~

```

```

11411 light-syntax,~
11412 light-syntax-expanded,~
11413 name,~
11414 no-cell-nodes,~
11415 nullify-dots,~
11416 pgf-node-code,~
11417 r,~
11418 renew-dots,~
11419 respect-arraystretch,~
11420 right-margin,~
11421 rounded-corners,~
11422 rules~(with~the~subkeys~'color'~and~'width'),~
11423 small,~
11424 t,~
11425 vlines,~
11426 xdots/color,~
11427 xdots/shorten-start(+),~
11428 xdots/shorten-end(+),~
11429 xdots/shorten(+)-and~
11430 xdots/line-style.
11431 }

11432 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
11433 {
11434   Unknown~key.\\
11435   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11436   \{NiceTabular\}. \\
11437   That~key~will~be~ignored. \\
11438   \c_@@_available_keys_str
11439 }
11440 {
11441   The~available~keys~are~(in~alphabetic~order):~
11442   &~in~blocks,~
11443   ampersand~in~blocks,~
11444   b,~
11445   baseline,~
11446   c,~
11447   caption,~
11448   cell-space-bottom-limit,~
11449   cell-space-limits,~
11450   cell-space-top-limit,~
11451   code-after,~
11452   code-for-first-col(+),~
11453   code-for-first-row(+),~
11454   code-for-last-col(+),~
11455   code-for-last-row(+),~
11456   columns-width,~
11457   corners,~
11458   custom-line,~
11459   create-blocks-in-col,~
11460   create-extra-nodes,~
11461   create-medium-nodes,~
11462   create-large-nodes,~
11463   extra-left-margin,~
11464   extra-right-margin,~
11465   first-col,~
11466   first-row,~
11467   hlines,~
11468   hvlines,~
11469   hvlines-except-borders,~
11470   label,~
11471   last-col,~
11472   last-row,~
11473   left-margin,~

```



```

11474 light-syntax,~
11475 light-syntax-expanded,~
11476 name,~
11477 no-cell-nodes,~
11478 notes~(several~subkeys),~
11479 nullify-dots,~
11480 pgf-node-code,~
11481 renew-dots,~
11482 respect-arraystretch,~
11483 right-margin,~
11484 rounded-corners,~
11485 rules~(with~the~subkeys~'color'~and~'width'),~
11486 short-caption,~
11487 t,~
11488 tabularnote,~
11489 vlines,~
11490 xdots/color,~
11491 xdots/shorten-start(+),~
11492 xdots/shorten-end(+),~
11493 xdots/shorten(+),~and~
11494 xdots/line-style.
11495 }
11496 \@@_msg_new:nnn { Duplicate-name }
11497 {
11498 Duplicate-name.\\
11499 The-name-' \l_keys_value_tl 'is-already-used-and-you-shouldn't-use~
11500 the-same-environment-name-twice.~You-can-go-on,~but,~
11501 maybe,~you-will-have-incorrect-results-especially~
11502 if-you-use-'columns-width=auto'.~If-you-don't-want-to-see-this~
11503 message-again,~use-the-key~'allow-duplicate-names'~in~
11504 ' \token_to_str:N \NiceMatrixOptions '.\\
11505 \bool_if:NF \g_@@_messages_for_Overleaf_bool
11506 { For-a-list-of-the-names-already-used,~type-H<return>. }
11507 }
11508 {
11509 The-names~already-defined-in~this~document~are:~
11510 \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
11511 }
11512 \@@_msg_new:nn { caption-above-in-env }
11513 {
11514 The-key~'caption-above'~must-be-used-in~\token_to_str:N \NiceMatrixOptions.\\
11515 That-key-will-be-ignored.
11516 }
11517 \@@_msg_new:nn { show-cell-names }
11518 {
11519 There-is-no-key~'show-cell-names'~in~nicematrix.\\
11520 You-should-use-the-command~\token_to_str:N \ShowCellNames\
11521 in~the~\token_to_str:N \CodeBefore\ or~the~\token_to_str:N
11522 \CodeAfter. \\
11523 That-key-will-be-ignored.
11524 }
11525 \@@_msg_new:nn { Option-auto-for-columns-width }
11526 {
11527 Erroneous~use.\\
11528 You-can't-give-the-value~'auto'~to~the-key~'columns-width'~here.~
11529 That-key-will-be-ignored.
11530 }
11531 \@@_msg_new:nn { NiceTabularX~without~X }
11532 {
11533 NiceTabularX~without~X.\\
11534 You-should-not-use~\{NiceTabularX\}~without~X~columns.\\
11535 However,~you-can-go-on.

```

```

11536     }
11537 \@@_msg_new:nn { Preamble~forgotten }
11538 {
11539     Preamble~forgotten.\\
11540     You~have~probably~forgotten~the~preamble~of~your~
11541     \@@_full_name_env: . \\
11542     This~error~is~fatal.
11543 }
11544 \@@_msg_new:nn { Invalid~col~number }
11545 {
11546     Invalid~column~number.\\
11547     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11548     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.\\
11549     Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
11550 }
11551 \@@_msg_new:nn { Invalid~row~number }
11552 {
11553     Invalid~row~number.\\
11554     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11555     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.\\
11556     Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
11557 }
11558 \@@_define_com:NNN p ( )
11559 \@@_define_com:NNN b [ ]
11560 \@@_define_com:NNN v | |
11561 \@@_define_com:NNN V \l \l
11562 \@@_define_com:NNN B \{ \}

```

Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	4
4	Parameters	9
5	The command <code>\tabularnote</code>	20
6	Command for creation of rectangle nodes	25
7	The options	26
8	Important code used by <code>{NiceArrayWithDelims}</code>	37
9	The <code>\CodeBefore</code>	52
10	The environment <code>{NiceArrayWithDelims}</code>	56
11	Construction of the preamble of the array	62
12	The redefinition of <code>\multicolumn</code>	79
13	The environment <code>{NiceMatrix}</code> and its variants	96
	13.1 Definition of <code>{pNiceMatrix}</code>	96
	13.2 The key <code>renew-matrix</code>	97
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	97
15	After the construction of the array	98
16	We draw the dotted lines	105
17	The actual instructions for drawing the dotted lines with <code>TikZ</code>	123
18	User commands available in the new environments	128
19	The command <code>\line</code> accessible in <code>\CodeAfter</code>	135
20	The command <code>\RowStyle</code>	136
21	Colors of cells, rows and columns	139
22	The vertical and horizontal rules	152
23	The empty corners	173
24	The environment <code>{NiceMatrixBlock}</code>	175
25	The extra nodes	176
26	The blocks	181
27	Automatic arrays	208
28	The redefinition of the command <code>\dotfill</code>	209
29	The command <code>\diagbox</code>	210

30	The keyword <code>\CodeAfter</code>	211
31	The delimiters in the preamble	212
32	The command <code>\SubMatrix</code>	213
33	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	222
34	The commands <code>HBrace</code> et <code>VBrace</code>	225
35	The command <code>TikzEveryCell</code>	228
36	The key <code>draw-trees-in-col</code>	230
37	The key <code>create-blocks-in-col</code>	231
38	The command <code>\ShowCellNames</code>	232
39	We process the options at package loading	234
40	About the package underscore	235
41	Compatibility with <code>threeparttable</code>	235
42	Error messages of the package	236