

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

July 22, 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 { 2025-06-01 }
17   { }
18 { \msg_critical:nn { nicematrix } { latex-too-old } }
```

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

```
19 \RequirePackage { amsmath }
```

*This document corresponds to the version 7.11a of `nicematrix`, at the date of 2026/07/22.

```

20 \msg_new:nnn { nicematrix } { array-too-old }
21 {
22   Your~version~of~the~package~'array'~is~too~old. \\
23   You~need~at~least~the~version~of~2025-09-25. \\
24   The~package~'nicematrix'~won't~be~loaded.
25 }
26 \RequirePackage{array}
27 \IfPackageAtLeastF { array }
28 { 2025/09/25 }
29 { \msg_critical:nn { nicematrix } { array-too-old} }

30 \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
31 \cs_new_protected:Npn \@@_warning:n { \msg_warning:nn { nicematrix } }
32 \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
33 \cs_generate_variant:Nn \@@_error:nn { n e }
34 \cs_new_protected:Npn \@@_error:nnnn { \msg_error:nnnn { nicematrix } }
35 \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
36 \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
37 \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).

```

38 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
39 {
40   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
41     { \msg_new:nnn { nicematrix } { #1 } { #2 } { #3 } }
42     {
43       \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).

```

44       \msg_new:nnn { nicematrix } { #1~+ } { #3 }
45     }
46 }

```

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

```

47 \cs_new_protected:Npn \@@_error_or_warning:n
48 {
49   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
50     \@@_warning:n
51     \@@_error:n
52 }

```

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”.

```

53 \bool_new:N \g_@@_messages_for_Overleaf_bool
54 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
55 {
56   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
57   || \str_if_eq_p:ee \c_sys_jobname_str { output } % for Overleaf
58 }

59 \@@_msg_new:nn { mdwtab-loaded }
60 {
61   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
62   This~error~is~fatal.
63 }

```

```

64 \hook_gput_code:nnn { begindocument / end } { . }
65 { \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.

```

66 \IfClassLoadedTF { revtex4-1 }
67 { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
68 {
69   \IfClassLoadedTF { revtex4-2 }
70   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
71   {

```

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

```

72     \cs_if_exist:NT \rvtx@ifformat@geq
73     { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
74     { \bool_const:Nn \c_@@_revtex_bool { \c_false_bool } }
75   }
76 }

77 \@@_msg_new:nn { class~revtex }
78 {
79   You~can't~use~this~version~of~'nicematrix'~in~a~class~of~REVTeX~because~
80   REVTeX~is~*not*~compatible~with~recent~versions~of~LaTeX.~Sorry...\\
81   The~package~'nicematrix'~won't~be~loaded.
82 }

83 \bool_if:NT \c_@@_revtex_bool
84 { \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`).

```

85 \cs_new_protected:Npn \@@_collect_options:n #1
86 {
87   \peek_meaning:NTF [
88     { \@@_collect_options:nw { #1 } }
89     { #1 { } }
90   }

```

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

```

91 \NewDocumentCommand \@@_collect_options:nw { m r[] }
92 { \@@_collect_options:nn { #1 } { #2 } }
93
94 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
95 {
96   \peek_meaning:NTF [
97     { \@@_collect_options:nnw { #1 } { #2 } }
98     { #1 { #2 } }
99   }

```

```

100
101 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
102 { \@@_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.

```

103 \tl_const:Nn \c_@@_c_tl { c }
104 \tl_const:Nn \c_@@_l_tl { l }
105 \tl_const:Nn \c_@@_r_tl { r }
106 \tl_const:Nn \c_@@_all_tl { all }
107 \tl_const:Nn \c_@@_dot_tl { . }
108 \str_const:Nn \c_@@_r_str { r }
109 \str_const:Nn \c_@@_c_str { c }
110 \str_const:Nn \c_@@_l_str { l }

111 \tl_const:Nn \c_@@_brace_tl { nicematrix/brace }
112 \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).

```

113 \tl_new:N \l_@@_argspec_tl

114 \cs_generate_variant:Nn \seq_set_split:Nnn { N o }
115 \cs_generate_variant:Nn \str_set:Nn { N o }
116 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
117 \prg_generate_conditional_variant:Nnn \clist_if_in:Nn { N e } { T , F, TF }
118 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { e } { T }
119 \prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
120 \cs_generate_variant:Nn \dim_min:nn { v }
121 \cs_generate_variant:Nn \dim_max:nn { v }

122 \AtBeginDocument
123 {
124   \IfPackageLoadedTF { tikz }
125   {

```

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.

```

126   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
127   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
128 }
129 {
130   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
131   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
132 }
133 }

```

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.

```

134 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
135 {
136   \iow_now:Nn \@mainaux
137   {
138     \ExplSyntaxOn
139     \cs_if_free:NT \pgfsyspdfmark
140     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
141     \ExplSyntaxOff
142   }
143   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
144 }

```

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.

```

145 \ProvideDocumentCommand { \iddots } { } {
146   {
147     \mathinner
148     {
149       \mkern 1 mu
150       \box_move_up:nn { 1 pt } { \hbox { . } }
151       \mkern 2 mu
152       \box_move_up:nn { 4 pt } { \hbox { . } }
153       \mkern 2 mu
154       \box_move_up:nn { 7 pt }
155       { \vbox:n { \kern 7 pt \hbox { . } } }
156       \mkern 1 mu
157     }
158   }

```

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.

```

159 \AtBeginDocument
160 {
161   \IfPackageLoadedT { booktabs }
162   {
163     \iow_now:Nn \@mainaux
164     {
165       \ExplSyntaxOn
166       \cs_if_exist_use:NT \nicematrix@redefine@check@rerun
167       \ExplSyntaxOff
168     }
169   }
170 }
171 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
172 {
173   \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`).

```

174   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
175   {

```

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

```

176     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } { 1 } { 3 } }
177     { \@@_old_pgful@check@rerun { ##1 } { ##2 } }
178   }
179 }

```

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`.

```

180 \AtBeginDocument
181 {
182   \cs_set_protected:Npe \@@_everycr:
183   {
184     \IfPackageLoadedTF { colortbl } \CT@everycr \everycr
185     { \noalign { \@@_in_everycr: } }
186   }
187   \IfPackageLoadedTF { colortbl }
188   {
189     \cs_new_eq:NN \@@_old_cellcolor: \cellcolor
190     \cs_new_eq:NN \@@_old_rowcolor: \rowcolor
191     \cs_new_protected:Npn \@@_revert_colortbl:
192     {
193       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
194       {
195         \cs_set_eq:NN \cellcolor \@@_old_cellcolor:
196         \cs_set_eq:NN \rowcolor \@@_old_rowcolor:
197       }
198     }
199   }

```

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).

```

199     \cs_new_protected:Npn \@@_replace_columncolor:
200     {
201       \tl_replace_all:Nnn \g_@@_array_preamble_tl
202       \columncolor
203       \@@_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.

```

204   }
205 }
206 {
207   \cs_new_protected:Npn \@@_revert_colortbl: { }
208   \cs_new_protected:Npn \@@_replace_columncolor:
209   { \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.

```

210   \def \CT@arc@ { }
211   \def \arrayrulecolor #1 # { \CT@arc { #1 } }
212   \def \CT@arc #1 #2
213   {
214     \dim_compare:nNnT \baselineskip = \c_zero_dim { \noalign }
215     { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
216   }

```

Idem for `\CT@drs@`.

```

217   \def \doublerulesepcolor #1 # { \CT@drs { #1 } }
218   \def \CT@drs #1 #2
219   {
220     \dim_compare:nNnT \baselineskip = \c_zero_dim { \noalign }
221     { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
222   }

```

```

223     \def \hline
224     {
225         \noalign { \ifnum 0 = ` } \fi
226         \cs_set_eq:NN \hskip \vskip
227         \cs_set_eq:NN \vrule \hrule
228         \cs_set_eq:NN \@width \@height
229         { \CT@arc@ \vline }
230         \futurelet \reserved@a
231         \@xhline
232     }
233 }
234 }

```

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.

```

235 \cs_set_nopar:Npn \@@_standard_cline: #1 { \@@_standard_cline:w #1 \q_stop }
236 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
237 {
238     \int_if_zero:nT \l_@@_first_col_int { \omit & }
239     \int_compare:nNnT { #1 } > 1
240     { \multispan { \int_eval:n { #1 - 1 } } & }
241     \multispan { \int_eval:n { #2 - #1 + 1 } }
242     {
243         \CT@arc@
244         \leaders \hrule \@height \arrayrulewidth \hfill

```

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

```

245     \skip_horizontal:N \c_zero_dim
246 }

```

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`.

```

247     \everycr { }
248     \cr
249     \noalign { \skip_vertical:n { - \arrayrulewidth } }
250 }

```

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.

```

251 \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`.

```

252 { \@@_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*.

```

253 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
254 \cs_generate_variant:Nn \@@_cline_i:nn { e }
255 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
256 {
257     \tl_if_empty:nTF { #3 }
258     { \@@_cline_iii:w #1|#2-#2 \q_stop }
259     { \@@_cline_ii:w #1|#2-#3 \q_stop }
260 }
261 \cs_set:Npn \@@_cline_ii:w #1|#2-#3- \q_stop
262 { \@@_cline_iii:w #1|#2-#3 \q_stop }

```

¹See question 99041 on TeX StackExchange.

```

263 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
264 {

```

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).

```

265   \int_compare:nNt { #1 } < { #2 }
266   { \multispan { \int_eval:n { #2 - #1 } } & }
267   \multispan { \int_eval:n { #3 - #2 + 1 } }
268   {
269     \CT@arc@
270     \leaders \hrule \@height \arrayrulewidth \hfill
271     \skip_horizontal:N \c_zero_dim
272   }

```

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

```

273   \peek_meaning_remove_ignore_spaces:NTF \cline
274   { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
275   { \everycr { } \cr }
276 }

```

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

```

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

278 \cs_new_protected:Npn \@@_set_CTarc:n #1
279 {
280   \tl_if_blank:nF { #1 }
281   {
282     \tl_if_head_eq_meaning:nNTF { #1 } [
283       { \def \CT@arc@ { \color #1 } }
284       { \def \CT@arc@ { \color { #1 } } }
285     ]
286   }
287 \cs_generate_variant:Nn \@@_set_CTarc:n { o }

288 \cs_new_protected:Npn \@@_set_CTdrsc:n #1
289 {
290   \tl_if_head_eq_meaning:nNTF { #1 } [
291     { \def \CT@drsc@ { \color #1 } }
292     { \def \CT@drsc@ { \color { #1 } } }
293   ]

```

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

```

294 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
295 {
296   \tl_if_head_eq_meaning:nNTF { #2 } [
297     { #1 #2 }
298     { #1 { #2 } }
299   ]
300 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```

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

```

301 \cs_new_protected:Npn \@@_color:n #1
302 { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }
303 \cs_generate_variant:Nn \@@_color:n { o }

304 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
305 {
306   \tl_set_rescan:Nno
307   #1

```



```

308     {
309         \char_set_catcode_other:N >
310         \char_set_catcode_other:N <
311     }
312     #1
313 }

```

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

```

314 \dim_new:N \l_@@_tmpc_dim
315 \dim_new:N \l_@@_tmpd_dim
316 \tl_new:N \l_@@_tmpc_tl
317 \tl_new:N \l_@@_tmpd_tl
318 \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.

```

319 \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).

```

320 \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.

```

321 \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.

```

322 \box_new:N \l_@@_the_array_box

```

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

```

323 \cs_new_protected:Npn \@@_qpoint:n #1
324 { \pgfpointanchor { \@@_env: - #1 } { center } }

```

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

```

325 \bool_new:N \l_@@_tabular_bool

```

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

```

326 \bool_new:N \g_@@_delims_bool
327 \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.

```
328 \bool_new:N \l_@@_preamble_bool
329 \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.

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

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

```
331 \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.

```
332 \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).

```
333 \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.).

```
334 \dim_new:N \l_@@_col_width_dim
335 \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.

```
336 \int_new:N \g_@@_row_total_int
337 \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).

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

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

```
339 \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.

```
340 \tl_new:N \l_@@_hpos_cell_tl
341 \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`.

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

Idem for the mono-row blocks.

```
343 \dim_new:N \g_@@_blocks_ht_dim
344 \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}`).

```
345 \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`.

```
346 \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.

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

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

```
348 \bool_new:N \l_@@_notes_detect_duplicates_bool
349 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

```
350 \bool_new:N \l_@@_initial_open_bool
351 \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.

```
352 \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).

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

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

```
354 \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).

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

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

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

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

```
357 \bool_new:N \g_@@_rotate_c_bool
```

The following boolean will be raised when the command `\rotate` is used with the key `-90`.

```
358 \bool_new:N \g_@@_rotate_minus_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.

```
359 \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`).

```
360 \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`.

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

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

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

```
363 \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 }`).

```
364 \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 it up).

```
365 \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 following sequence that will contain information about the size of the array.

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

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

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

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

```
369 \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).

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

For `\multicolumn`.

```
371 \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`.

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

```
373 \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 `..`.

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

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

```
376 \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`).

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

```

378 \cs_new_protected:Npn \@@_test_if_math_mode:
379 {
380   \if_mode_math: \else:
381     \@@_fatal:n { Outside~math~mode }
382   \fi:
383 }

```

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.

```

384 \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”.

```

385 \colorlet { nicematrix-last-col } { . }
386 \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*).

```

387 \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*.

```

388 \str_new:N \g_@@_com_or_env_str
389 \str_gset:Nn \g_@@_com_or_env_str { environment }

```

```

390 \bool_new:N \l_@@_bold_row_style_bool

```

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

```

391 \clist_new:N \g_@@_cbic_clist

```

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

```

392 \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)`.

```

393 \cs_new:Npn \@@_full_name_env:
394 {
395   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
396     { command \space \c_backslash_str \g_@@_name_env_str }
397     { environment \space \{ \g_@@_name_env_str \} }
398 }

```

```

399 \tl_new:N \g_@@_cell_after_hook_tl

```

The argument is given by curryfication.

```

400 \cs_new_protected:Npn \@@_put_in_cell_after_hook:n
401 { \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.

```

402 \tl_new:N \g_@@_pre_code_before_tl
403 \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`.

```
404 \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.

```
405 \tl_new:N \g_@@_pre_code_after_tl
406 \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.

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

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

```
408 \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.

```
409 \int_new:N \l_@@_old_iRow_int
410 \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).

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

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

```
412 \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.

```
413 \bool_new:N \l_@@_X_columns_aux_bool
414 \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.

```
415 \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).

```
416 \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).

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

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

```
419 \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).

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

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

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

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

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

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

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

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

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

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

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

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

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

```
431 \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”.

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

```
433 \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.

```
434 \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!).

```
435 \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.

```
436 \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}`.

```
437 \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.

```
438 \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`).

```
439 \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`).

```
440 \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.

```
441 \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).

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

```
443 \seq_new:N \g_@@_multicolumn_sizes_seq
```

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

```
444 \int_new:N \l_@@_multiplicity_int
```

```
445 \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.

```
446 \int_new:N \g_@@_ddots_int
```

```
447 \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.

```
448 \dim_new:N \g_@@_delta_x_one_dim
```

```
449 \dim_new:N \g_@@_delta_y_one_dim
```

```
450 \dim_new:N \g_@@_delta_x_two_dim
```

```
451 \dim_new:N \g_@@_delta_y_two_dim
```

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

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).

```

452 \int_new:N \l_@@_row_min_int
453 \int_new:N \l_@@_row_max_int
454 \int_new:N \l_@@_col_min_int
455 \int_new:N \l_@@_col_max_int

456 \int_new:N \l_@@_initial_i_int
457 \int_new:N \l_@@_initial_j_int
458 \int_new:N \l_@@_final_i_int
459 \int_new:N \l_@@_final_j_int

```

The following counters will be used when drawing the rules.

```

460 \int_new:N \l_@@_segment_start_int
461 \int_new:N \l_@@_segment_end_int

```

The following sequence will be used when the command `\SubMatrix` is used in the `\CodeBefore` (and not in the `\CodeAfter`). It will contain the position of all the sub-matrices specified in the `\CodeBefore`. Each sub-matrix is represented by an “object” of the form $\{i\}\{j\}\{k\}\{l\}$ where i and j are the number of row and column of the upper-left cell and k and l the number of row and column of the lower-right cell.

```

462 \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).

```

463 \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`.

```

464 \tl_new:N \l_@@_draw_tl
465 \seq_new:N \l_@@_tikz_seq
466 \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`).

```

467 \str_new:N \l_@@_hpos_block_str
468 \str_set:Nn \l_@@_hpos_block_str { c }
469 \bool_new:N \l_@@_hpos_of_block_cap_bool
470 \bool_new:N \l_@@_p_block_bool

471 \bool_new:N \l_@@_fix_vertex_bool

```

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

```

472 \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).

```

473 \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.

```

474 \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}`).

```

475 \bool_new:N \l_@@_hvlines_bool

```

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

```
476 \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`).

```
477 \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.

```
478 \int_new:N \l_@@_first_row_int
479 \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.

```
480 \int_new:N \l_@@_first_col_int
481 \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).

```
482 \int_new:N \l_@@_last_row_int
483 \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

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

Idem for `\l_@@_last_col_without_value_bool`

```
485 \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.

```
486 \int_new:N \l_@@_last_col_int
487 \int_set:Nn \l_@@_last_col_int { -2 }
```

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.

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

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

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

```
488 \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`).

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

Some utilities

```
490 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
491 {
```

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

```
492 \def \l_tmpa_tl { #1 }
493 \def \l_tmpb_tl { #2 }
494 }
```

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`.

```
495 \cs_new_protected:Npn \@@_expand_clist_hvlines:NN #1 #2
496 {
497 \clist_if_in:NnF #1 { all }
498 {
499 \clist_clear:N \l_tmpa_clist
500 \clist_map_inline:Nn #1
501 {
502 \tl_if_head_eq_meaning:nNTF { ##1 } -
503 {
```

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.

```
504 \int_if_zero:nF { #2 }
505 {
506 \clist_put_right:Ne \l_tmpa_clist
507 { \int_eval:n { #2 + (##1) + 1 } }
508 }
509 }
510 {
```

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

```
511 \tl_if_in:nnTF { ##1 } { - }
512 { \@@_cut_on_hyphen:w ##1 \q_stop }
513 {
```

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

```
514 \def \l_tmpa_tl { ##1 }
515 \def \l_tmpb_tl { ##1 }
516 }
517 \step_inline:nnn \l_tmpa_tl \l_tmpb_tl
518 { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
```

```

519         }
520     }
521     \tl_set_eq:NN #1 \l_tmpa_clist
522 }
523 }

```

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.

```

524 \AtBeginDocument
525 {
526     \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
527     \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
528 }

```

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.

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

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.

```
529 \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`.

```
530 \int_new:N \g_@@_tabularnote_int
531 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }

532 \seq_new:N \g_@@_notes_seq
533 \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.

```
534 \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.

```
535 \seq_new:N \l_@@_notes_labels_seq
536 \newcounter { nicematrix_draft }
537 \cs_new_protected:Npn \@@_notes_format:n #1
538 {
539   \setcounter { nicematrix_draft } { #1 }
540   \@@_notes_style:n { nicematrix_draft }
541 }
```

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

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

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

```
543 \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`.

```
544 \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`.

```
545 \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`).

```
546 \AtBeginDocument
547 {
548   \IfPackageLoadedTF { enumitem }
549   {
```

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.

```

550     \newlist { tabularnotes } { enumerate } { 1 }
551     \setlist [ tabularnotes ]
552     {
553         topsep = \c_zero_dim ,
554         noitemsep ,
555         leftmargin = * ,
556         align = left ,
557         labelsep = \c_zero_dim ,
558         label =
559             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
560     }
561     \newlist { tabularnotes* } { enumerate* } { 1 }
562     \setlist [ tabularnotes* ]
563     {
564         afterlabel = \nobreak ,
565         itemjoin = \quad ,
566         label =
567             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
568     }

```

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`.

```

569     \NewDocumentCommand { \tabularnote } { o m }
570     {
571         \bool_lazy_or:nnT \l_@@_in_env_bool { \cs_if_exist_p:N \@capytype }
572         {
573             \bool_lazy_and:nnTF \l_@@_in_env_bool { ! \l_@@_tabular_bool }
574             { \@@_error:n { tabularnote~forbidden } }
575             {

```

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

```

576         \bool_if:NTF \l_@@_in_caption_bool
577         {
578             \tl_if_novalue:nTF { #1 }
579             { \@@_tabularnote_caption:nn { #1 } }
580             { \@@_tabularnote_caption:nn { \exp_not:n { #1 } } }
581         }
582         {
583             \tl_if_novalue:nTF { #1 }
584             { \@@_tabularnote:nn { #1 } }
585             { \@@_tabularnote:nn { \exp_not:n { #1 } } }
586         }
587         { #2 }
588     }
589 }
590 }
591 }
592 {
593     \NewDocumentCommand \tabularnote { o m }
594     { \@@_err_enumitem_not_loaded: }
595 }
596 }
597 \cs_new_protected:Npn \@@_err_enumitem_not_loaded:
598 {
599     \@@_error_or_warning:n { enumitem~not~loaded }
600     \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
601 }

```

```

602 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
603 { \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`.

```

604 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
605 {

```

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).

```

606 \int_zero:N \l_tmpa_int
607 \bool_if:NT \l_@@_notes_detect_duplicates_bool
608 {

```

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

{label}{text of the tabularnote}.

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

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

```

609 \int_zero:N \l_tmpb_int
610 \seq_map_indexed_inline:Nn \g_@@_notes_seq
611 {
612 \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
613 \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
614 {
615 \tl_if_novalue:nTF { #1 }
616 { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
617 { \int_set:Nn \l_tmpa_int { ##1 } }
618 \seq_map_break:
619 }
620 }
621 \int_if_zero:nF \l_tmpa_int
622 { \int_add:Nn \l_tmpa_int { \g_@@_notes_caption_int } }
623 }
624 \int_if_zero:nT \l_tmpa_int
625 {
626 \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
627 \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
628 }
629 \seq_put_right:Ne \l_@@_notes_labels_seq
630 {
631 \tl_if_novalue:nTF { #1 }
632 {
633 \@@_notes_format:n
634 {
635 \int_eval:n
636 {
637 \int_if_zero:nTF \l_tmpa_int
638 \c@tabularnote
639 \l_tmpa_int
640 }
641 }
642 }
643 { #1 }
644 }
645 \peek_meaning:NF \tabularnote
646 {

```

If the following token is *not* a `\tabularnote`, we have finished the sequence of successive commands `\tabularnote` and we have to format the labels of these tabular notes (in the array). We compose those labels in a box `\l_tmpa_box` because we will do a special construction in order to have this box in an overlapping position if we are at the end of a cell when `\l_@@_hpos_cell_tl` is equal to `c` or `r`.

```
647 \hbox_set:Nn \l_tmpa_box
648 {
```

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

```
649 \@@_notes_label_in_tabular:n
650 { \seq_use:Nn \l_@@_notes_labels_seq { , } }
651 }
```

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

```
652 \int_gdecr:N \c@tabularnote
653 \int_set_eq:NN \l_tmpa_int \c@tabularnote
```

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

```
654 \int_gincr:N \g_@@_tabularnote_int
655 \refstepcounter { tabularnote }
656 \int_compare:nNnT \l_tmpa_int = \c@tabularnote
657 { \int_gincr:N \c@tabularnote }
658 \seq_clear:N \l_@@_notes_labels_seq
659 \bool_lazy_or:nnTF
660 { \str_if_eq_p:ee c \l_@@_hpos_cell_tl }
661 { \str_if_eq_p:ee r \l_@@_hpos_cell_tl }
662 {
663 \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).

```
664 \skip_horizontal:n { \box_wd:N \l_tmpa_box }
665 }
666 { \box_use:N \l_tmpa_box }
667 }
668 }
```

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`.

```
669 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
670 {
671 \bool_if:NTF \g_@@_caption_finished_bool
672 {
673 \int_compare:nNnT \c@tabularnote = \g_@@_notes_caption_int
674 { \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`!

```
675 \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
676 { \@@_error:n { Identical~notes~in~caption } }
677 }
678 {
```

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

```
679 \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
680 {
```


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.

```

681         \bool_gset_true:N \g_@@_caption_finished_bool
682         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
683         \int_gzero:N \c@tabularnote
684     }
685     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
686 }

```

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!

```

687     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
688     \seq_put_right:Ne \l_@@_notes_labels_seq
689     {
690         \tl_if_novalue:nTF { #1 }
691         { \@@_notes_format:n { \int_use:N \c@tabularnote } }
692         { #1 }
693     }
694     \peek_meaning:NF \tabularnote
695     {
696         \@@_notes_label_in_tabular:n { \seq_use:Nn \l_@@_notes_labels_seq { , } }
697         \seq_clear:N \l_@@_notes_labels_seq
698     }
699 }

700 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
701 { \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.

```

702 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
703 {
704     \begin { pgfscope }
705     \pgfset
706     {
707         inner~sep = \c_zero_dim ,
708         minimum~size = \c_zero_dim
709     }
710     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
711     \pgfnode
712     { rectangle }
713     { center }
714     {
715         \vbox_to_ht:nn
716         { \dim_abs:n { #5 - #3 } }
717         {
718             \vfill
719             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
720         }
721     }
722     { #1 }
723     { }
724     \end { pgfscope }
725 }

```

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.

```

726 \cs_new_protected:Npn \@@pgf_rect_node:nnn #1 #2 #3
727 {
728   \begin { pgfscope }
729   \pgfset
730   {
731     inner~sep = \c_zero_dim ,
732     minimum~size = \c_zero_dim
733   }
734   \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
735   \pgfpointdiff { #3 } { #2 }
736   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
737   \pgfnode
738   { rectangle }
739   { center }
740   {
741     \vbox_to_ht:nn
742     { \dim_abs:n \l_tmpb_dim }
743     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
744   }
745   { #1 }
746   { }
747   \end { pgfscope }
748 }

```

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).

```

749 \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.

```

750 \dim_new:N \l_@@_xdots_inter_dim
751 \AtBeginDocument
752 { \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).

```

753 \dim_new:N \l_@@_xdots_shorten_start_dim
754 \dim_new:N \l_@@_xdots_shorten_end_dim
755 \AtBeginDocument
756 {
757   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
758   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
759 }

```

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.

```

760 \dim_new:N \l_@@_xdots_radius_dim
761 \AtBeginDocument
762 { \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.

```

763 \tl_new:N \l_@@_xdots_line_style_tl
764 \tl_const:Nn \c_@@_standard_tl { standard }
765 \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 option `light-syntax-expanded`.

```

766 \bool_new:N \l_@@_light_syntax_bool
767 \bool_new:N \l_@@_light_syntax_expanded_bool

```

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

```

768 \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).

```

769 \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!

```

770 \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.

```

771 \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”.

```

772 \bool_new:N \l_@@_medium_nodes_bool
773 \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).

```

774 \bool_new:N \l_@@_except_borders_bool

```

The following parameter corresponds to the key `width-of-false`.

```

775 \dim_new:N \l_@@_width_of_false_dim
776 \dim_set:Nn \l_@@_width_of_false_dim { 2 pt }

```

```

777 \keys_define:nn { nicematrix / xdots }
778 {

```

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.).

```

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

```

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

```

833   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
834   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,

```

```
835 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).

```
836 draw-first .code:n = \prg_do_nothing: ,
837 unknown .code:n =
838   \@@_unknown_key:nn { nicematrix / xdots } { Unknown~key~for~xdots }
839 }
```

```
840 \keys_define:nn { nicematrix / trees }
841 {
842   color.tl_set:N = \l_@@_trees_color_tl ,
843   color .value_required:n = true ,
844   line-width .dim_set:N = \l_@@_trees_line_width_dim ,
845   rounded-corners .dim_set:N = \l_@@_trees_rounded_corners_dim ,
846   rounded-corners .default:n = 2 pt ,
847   unknown .code:n =
848     \@@_unknown_key:nn { nicematrix / trees } { Unknown~key~for~trees }
849 }
```

```
850 \keys_define:nn { nicematrix / rules }
851 {
852   fix-vertex .code:n =
853     \bool_set_true:N \l_@@_fix_vertex_bool
854     \socket_assign_plug:nn { nicematrix / draw-at-odd-vertex-h } { active }
855     \socket_assign_plug:nn { nicematrix / draw-at-odd-vertex-v } { active } ,
856   color .tl_set:N = \l_@@_rules_color_tl ,
857   color .value_required:n = true ,
858   width .dim_set:N = \arrayrulewidth ,
859   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
860 }
```

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

```
861 \keys_define:nn { nicematrix / Global }
862 {
863   width-of-false .dim_set:N = \l_@@_width_of_false_dim ,
864   width-of-false .value_required:n = true ,
865   width-of-false+ .code:n =
866     \dim_add:Nn \l_@@_width_of_false_dim { #1 } ,
867   width-of-false+ .value_required:n = true ,
868   width-of-false~+ .meta:n = { width-of-false+ = #1 } ,
869   fix-vertex .value_forbidden:n = true ,
870   brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
871   brace-shift+ .code:n =
872     \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
873   brace-shift+ .value_required:n = true ,
874   brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
875   caption-above .code:n = \@@_error_or_warning:n { caption-above~in~env } ,
876   show-cell-names .code:n = \@@_error_or_warning:n { show-cell-names } ,
877   color-inside .code:n = \@@_fatal:n { key~color~inside } ,
878   colortbl-like .code:n = \@@_fatal:n { key~color~inside } ,
879   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
880   &-in-blocks .meta:n = ampersand-in-blocks ,
881   no-cell-nodes .choices:nn = { true , false }
882   {
883     \tl_if_eq:NnTF \l_keys_choice_tl { true }
884     {
885       \bool_set_true:N \l_@@_no_cell_nodes_bool
886       \cs_set_eq:NN \@@_node_cell: \@@_node_cell_inactive:
```

```

887     }
888     {
889         \bool_set_false:N \l_@@_no_cell_nodes_bool
890         \cs_set_eq:NN \@@_node_cell: \@@_node_cell_active:
891     }
892 } ,
893 no-cell-nodes .default: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.

```

894 rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
895 rounded-corners .default:n = 4 pt ,
896 custom-line .code:n = \@@_custom_line:n { #1 } ,
897 default-line .code:n = \@@_default_line:n { #1 } ,
898 rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
899 rules .value_required:n = true ,
900 trees .code:n = \keys_set:nn { nicematrix / trees } { #1 } ,
901 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`.

```

902 standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
903 cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
904 cell-space-top-limit+ .code:n =
905     \dim_add:Nn \l_@@_cell_space_top_limit_dim { #1 } ,
906 cell-space-top-limit+ .value_required:n = true ,
907 cell-space-top-limit+ .meta:n = { cell-space-top-limit+ = #1 } ,
908 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
909 cell-space-bottom-limit+ .code:n =
910     \dim_add:Nn \l_@@_cell_space_bottom_limit_dim { #1 } ,
911 cell-space-bottom-limit+ .value_required:n = true ,
912 cell-space-bottom-limit+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
913 cell-space-limits .meta:n =
914     {
915         cell-space-top-limit = #1 ,
916         cell-space-bottom-limit = #1 ,
917     } ,
918 cell-space-limits .value_required:n = true ,
919 cell-space-limits+ .meta:n =
920     {
921         cell-space-top-limit += #1 ,
922         cell-space-bottom-limit += #1 ,
923     } ,
924 cell-space-limits+ .value_required:n = true ,
925 cell-space-limits+ .meta:n = { cell-space-limits+ = #1 } ,
926 xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
927 light-syntax .choices:nn = { true , false }
928     {
929         \tl_if_eq:VnTF \l_keys_value_tl { true }
930             { \bool_set_true:N \l_@@_light_syntax_bool }
931             { \bool_set_false:N \l_@@_light_syntax_bool }
932         \bool_set_false:N \l_@@_light_syntax_expanded_bool
933     } ,
934 light-syntax .default:n = true ,
935 light-syntax-expanded .choices:nn = { true , false }
936     {
937         \tl_if_eq:VnTF \l_keys_value_tl { true }
938             {
939                 \bool_set_true:N \l_@@_light_syntax_bool
940                 \bool_set_true:N \l_@@_light_syntax_expanded_bool
941             }
942             {
943                 \bool_set_false:N \l_@@_light_syntax_bool

```

```

944         \bool_set_false:N \l_@@_light_syntax_expanded_bool
945     }
946 } ,
947 light-syntax-expanded .default:n = true ,

```

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

```

948 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
949 end-of-row .value_required:n = true ,
950 end-of-row .initial:n = ; ,
951
952 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
953 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
954 last-row .int_set:N = \l_@@_last_row_int ,
955 last-row .default:n = -1 ,
956
957 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
958 code-for-first-col .value_required:n = true ,
959 code-for-first-col+ .code:n =
960     { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
961 code-for-first-col+ .value_required:n = true ,
962 code-for-first-col+ .meta:n = { code-for-first-col+ = #1 } ,
963
964 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
965 code-for-last-col .value_required:n = true ,
966 code-for-last-col+ .code:n =
967     { \tl_put_right:Nn \l_@@_code_for_last_col_tl { #1 } } ,
968 code-for-last-col+ .value_required:n = true ,
969 code-for-last-col+ .meta:n = { code-for-last-col+ = #1 } ,
970
971 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
972 code-for-first-row .value_required:n = true ,
973 code-for-first-row+ .code:n =
974     { \tl_put_right:Nn \l_@@_code_for_first_row_tl { #1 } } ,
975 code-for-first-row+ .value_required:n = true ,
976 code-for-first-row+ .meta:n = { code-for-first-row+ = #1 } ,
977
978 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
979 code-for-last-row .value_required:n = true ,
980 code-for-last-row+ .code:n =
981     { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
982 code-for-last-row+ .value_required:n = true ,
983 code-for-last-row+ .meta:n = { code-for-last-row+ = #1 } ,
984
985 hlines .clist_set:N = \l_@@_hlines_clist ,
986 hlines .default:n = all ,
987 vlines .clist_set:N = \l_@@_vlines_clist ,
988 vlines .default:n = all ,
989
990 vlines-in-sub-matrix .code:n =
991     {
992         \tl_if_single_token:nTF { #1 }
993         {
994             \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
995             { \@@_error:nn { Forbidden~letter } { #1 } }

```

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

```

996         { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
997     }
998     { \@@_error:n { One~letter~allowed } }
999 } ,
1000 vlines-in-sub-matrix .value_required:n = true ,
1001 hvlines .code:n =
1002     {

```

```

1003     \bool_set_true:N \l_@@_hvlines_bool
1004     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
1005     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
1006   } ,
1007   hvlines .value_forbidden:n = true ,
1008   hvlines-except-borders .code:n =
1009   {
1010     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
1011     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
1012     \bool_set_true:N \l_@@_hvlines_bool
1013     \bool_set_true:N \l_@@_except_borders_bool
1014   } ,
1015   hlines-except-borders .value_forbidden:n = true ,
1016   hlines-except-borders .code:n =
1017   {
1018     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
1019     \bool_set_true:N \l_@@_hlines_bool
1020     \bool_set_true:N \l_@@_except_borders_bool
1021   } ,
1022   hlines-except-borders .value_forbidden:n = true ,
1023   parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,
1024   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.

```

1025   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
1026   renew-dots .value_forbidden:n = true ,
1027   nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
1028   create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
1029   create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
1030   create-extra-nodes .meta:n =
1031   { create-medium-nodes , create-large-nodes } ,
1032   left-margin .dim_set:N = \l_@@_left_margin_dim ,
1033   left-margin .default:n = \arraycolsep ,
1034   right-margin .dim_set:N = \l_@@_right_margin_dim ,
1035   right-margin .default:n = \arraycolsep ,
1036   margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
1037   margin .default:n = \arraycolsep ,
1038   extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
1039   extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
1040   extra-margin .meta:n =
1041   { extra-left-margin = #1 , extra-right-margin = #1 } ,
1042   extra-margin .value_required:n = true ,
1043   respect-arraystretch .code:n =
1044   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
1045   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.

```

1046   pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
1047   pgf-node-code .value_required:n = true ,
1048 }

```

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

```

1049 \keys_define:nn { nicematrix / environments }
1050 {
1051   draw-trees-in-col .code:n =
1052   \clist_gput_right:Nn \g_@@_col_with_trees_clist { #1 } ,
1053   draw-trees-in-col .value_required:n = true ,
1054   create-blocks-in-col .code:n =
1055   \clist_gput_right:Nn \g_@@_cbic_clist { #1 } ,

```



```

1056   create-blocks-in-col .value_required:n = true ,
1057   corners .clist_set:N = \l_@@_corners_clist ,
1058   corners .default:n = { NW , SW , NE , SE } ,
1059   code-before .code:n =
1060   {
1061     \tl_if_empty:nF { #1 }
1062     {
1063       \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1064       \bool_set_true:N \l_@@_code_before_bool
1065     }
1066   } ,
1067   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}`.

```

1068   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
1069   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
1070   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).

```

1071   baseline .tl_set:N = \l_@@_baseline_tl ,
1072   baseline .value_required:n = true ,
1073   baseline .initial:n = c ,
1074   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)`.

```

1075   \str_if_eq:eeTF { #1 } { auto }
1076   { \bool_set_true:N \l_@@_auto_columns_width_bool }
1077   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
1078   columns-width .value_required:n = true ,
1079   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.

```

1080   \legacy_if:nF { measuring@ }
1081   {
1082     \str_set:Ne \l_@@_name_str { #1 }
1083     \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
1084     { \@@_err_duplicate_names:n { #1 } }
1085     { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
1086   } ,
1087   name .value_required:n = true ,
1088   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
1089   code-after .value_required:n = true ,
1090 }

1091 \cs_set:Npn \@@_err_duplicate_names:n #1
1092 { \@@_error:nn { Duplicate-name } { #1 } }

1093 \keys_define:nn { nicematrix / notes }
1094 {
1095   no-print .bool_set:N = \l_@@_notes_no_print_bool ,
1096   para .bool_set:N = \l_@@_notes_para_bool ,
1097   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
1098   code-before .value_required:n = true ,
1099   code-before+ .code:n =
1100   \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1101   code-before+ .value_required:n = true ,
1102   code-before~+ .code:n =
1103   \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1104   code-before~+ .value_required:n = true ,
1105   code-after .tl_set:N = \l_@@_notes_code_after_tl ,

```

```

1106 code-after .value_required:n = true ,
1107 bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
1108 style .cs_set:Np = \@@_notes_style:n #1 ,
1109 style .value_required:n = true ,
1110 label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
1111 label-in-tabular .value_required:n = true ,
1112 label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1113 label-in-list .value_required:n = true ,
1114 enumitem-keys .code:n =
1115 {
1116   \AtBeginDocument
1117   {
1118     \IfPackageLoadedT { enumitem }
1119     { \setlist* [ tabularnotes ] { #1 } }
1120   }
1121 } ,
1122 enumitem-keys .value_required:n = true ,
1123 enumitem-keys-para .code:n =
1124 {
1125   \AtBeginDocument
1126   {
1127     \IfPackageLoadedT { enumitem }
1128     { \setlist* [ tabularnotes* ] { #1 } }
1129   }
1130 } ,
1131 enumitem-keys-para .value_required:n = true ,
1132 detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1133 unknown .code:n =
1134   \@@_unknown_key:nn { nicematrix / notes } { Unknown~key~for~notes }
1135 }

```

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.

```

1136 \keys_define:nn { nicematrix / delimiters }
1137 {
1138   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1139   color .tl_set:N = \l_@@_delimiters_color_tl ,
1140   color .value_required:n = true ,
1141 }

```

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

```

1142 \keys_define:nn { nicematrix }
1143 {
1144   NiceMatrixOptions .inherit:n =
1145     { nicematrix / Global } ,
1146   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1147   NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1148   NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1149   NiceMatrixOptions / trees .inherit:n = nicematrix / trees ,
1150   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1151   SubMatrix / rules .inherit:n = nicematrix / rules ,
1152   CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1153   CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1154   CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1155   NiceMatrix .inherit:n =
1156   {

```

```

1157     nicematrix / Global ,
1158     nicematrix / environments ,
1159   } ,
1160   NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1161   NiceMatrix / rules .inherit:n = nicematrix / rules ,
1162   NiceMatrix / trees .inherit:n = nicematrix / trees ,
1163   NiceTabular .inherit:n =
1164   {
1165     nicematrix / Global ,
1166     nicematrix / environments
1167   } ,
1168   NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1169   NiceTabular / rules .inherit:n = nicematrix / rules ,
1170   NiceTabular / notes .inherit:n = nicematrix / notes ,
1171   NiceTabular / trees .inherit:n = nicematrix / trees ,
1172   NiceArray .inherit:n =
1173   {
1174     nicematrix / Global ,
1175     nicematrix / environments ,
1176   } ,
1177   NiceArray / xdots .inherit:n = nicematrix / xdots ,
1178   NiceArray / rules .inherit:n = nicematrix / rules ,
1179   NiceArray / trees .inherit:n = nicematrix / trees ,
1180   pNiceArray .inherit:n =
1181   {
1182     nicematrix / Global ,
1183     nicematrix / environments ,
1184   } ,
1185   pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1186   pNiceArray / rules .inherit:n = nicematrix / rules ,
1187   pNiceArray / trees .inherit:n = nicematrix / trees
1188 }

```

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

```

1189 \keys_define:nn { nicematrix / NiceMatrixOptions }
1190 {
1191   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1192   delimiters / color .value_required:n = true ,
1193   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1194   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1195   delimiters .value_required:n = true ,
1196   width .dim_set:N = \l_@@_width_dim ,
1197   last-col .code:n =
1198     \tl_if_empty:nF { #1 }
1199     { \@@_error:n { last-col-non-empty~for~NiceMatrixOptions } }
1200     \int_zero:N \l_@@_last_col_int ,
1201   small .bool_set:N = \l_@@_small_bool ,
1202   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.

```

1203   renew-matrix .code:n = \@@_renew_matrix: ,
1204   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}`.

```

1205   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.

```

1206   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)`.

```

1207     \str_if_eq:eeTF { #1 } { auto }
1208     { \@@_error:n { Option~auto~for~columns~width } }
1209     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1210     allow-duplicate-names .code:n =
1211     \cs_set:Nn \@@_err_duplicate_names:n { } ,
1212     allow-duplicate-names .value_forbidden:n = true ,
1213     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1214     notes .value_required:n = true ,
1215     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1216     sub-matrix .value_required:n = true ,
1217     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1218     matrix / columns-type .value_required:n = true ,
1219     caption-above .bool_set:N = \l_@@_caption_above_bool ,
1220     unknown .code:n =
1221     \@@_unknown_key:nn
1222     { nicematrix / Global , nicematrix / NiceMatrixOptions }
1223     { Unknown-key-for~NiceMatrixOptions }
1224 }

```

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

```

1225 \NewDocumentCommand \NiceMatrixOptions { m }
1226 { \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.

```

1227 \keys_define:nn { nicematrix / NiceMatrix }
1228 {
1229     last-col .code:n = \tl_if_empty:nTF { #1 }
1230     {
1231         \bool_set_true:N \l_@@_last_col_without_value_bool
1232         \int_set:Nn \l_@@_last_col_int { -1 }
1233     }
1234     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1235     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1236     columns-type .value_required:n = true ,
1237     l .meta:n = { columns-type = l } ,
1238     r .meta:n = { columns-type = r } ,
1239     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1240     delimiters / color .value_required:n = true ,
1241     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1242     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1243     delimiters .value_required:n = true ,
1244     small .bool_set:N = \l_@@_small_bool ,
1245     small .value_forbidden:n = true ,
1246     unknown .code:n =
1247     \@@_unknown_key:nn
1248     { nicematrix / Global , nicematrix / environments , nicematrix / NiceMatrix }
1249     { Unknown-key-for~NiceMatrix }
1250 }

```

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

```

1251 \keys_define:nn { nicematrix / NiceArray }
1252 {

```

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.

```

1253     small .bool_set:N = \l_@@_small_bool ,
1254     small .value_forbidden: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 / NiceArray , nicematrix / Global , nicematrix / environments}
1263         { Unknown-key-for-NiceArray }
1264 }
1265 \keys_define:nn { nicematrix / pNiceArray }
1266 {
1267     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1268     last-col .code:n = \tl_if_empty:nF { #1 }
1269         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1270         \int_zero:N \l_@@_last_col_int ,
1271     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1272     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1273     delimiters / color .value_required:n = true ,
1274     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1275     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1276     delimiters .value_required:n = true ,
1277     small .bool_set:N = \l_@@_small_bool ,
1278     small .value_forbidden:n = true ,
1279     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1280     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1281     unknown .code:n =
1282         \@@_unknown_key:nn
1283         { nicematrix / pNiceArray , nicematrix / Global , nicematrix / environments }
1284         { Unknown-key-for-NiceMatrix }
1285 }

```

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

```

1286 \keys_define:nn { nicematrix / NiceTabular }
1287 {

```

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.

```

1288     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1289         \bool_set_true:N \l_@@_width_used_bool ,
1290     width .value_required:n = true ,
1291     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1292     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1293     tabularnote .value_required:n = true ,
1294     caption .tl_set:N = \l_@@_caption_tl ,
1295     caption .value_required:n = true ,
1296     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1297     short-caption .value_required:n = true ,
1298     label .tl_set:N = \l_@@_label_tl ,
1299     label .value_required:n = true ,
1300     last-col .code:n = \tl_if_empty:nF { #1 }
1301         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1302         \int_zero:N \l_@@_last_col_int ,
1303     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1304     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1305     unknown .code:n =
1306         \@@_unknown_key:nn

```

```

1307      { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
1308      { Unknown-key-for-NiceTabular }
1309    }

```

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
```

```

1310 \keys_define:nn { nicematrix / CodeAfter }
1311 {
1312   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1313   delimiters / color .value_required:n = true ,
1314   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1315   rules .value_required:n = true ,
1316   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1317   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1318   sub-matrix .value_required:n = true ,
1319   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1320 }

```

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}`).

```

1321 \cs_new_protected:Npn \@@_cell_begin:
1322 {

```

`\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.

```
1323   \tl_gclear:N \g_@@_cell_after_hook_tl
```

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

```
1324   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```

1325   \cs_set_eq:NN \Hline \@@_Hline_in_cell:
1326   \cs_set_eq:NN \FalseRow \@@_FalseRow_in_cell: % added 2026-07-17

```

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

```
1327   \int_gincr:N \c@jCol
```

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

Here is a version with the standard syntax of L3.

```

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

```

We will use a version a little more efficient.

```

1328   \if_int_compare:w \c@jCol = \c_one_int
1329     \if_int_compare:w \l_@@_first_col_int = \c_one_int
1330     \@@_begin_of_row:
1331   \fi:
1332 \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:`.

```
1333 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1334 \@@_tuning_not_tabular_begin:
1335 \@@_tuning_exterior_rows:
1336 \g_@@_row_style_tl
1337 }
1338 \cs_new_protected:Npn \@@_tuning_exterior_rows: { }
```

Here is a version with the standard syntax of the L3 programming layer.

```
\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.

```
1339 \cs_new_protected:Npn \@@_tuning_first_last_row:
1340 {
1341   \if_int_compare:w \c@iRow = \c_zero_int
1342   \if_int_compare:w \c@jCol > \c_zero_int
1343   \l_@@_code_for_first_row_tl
1344   \@@_tuning_key_small:
1345   \xglobal \colorlet { nicematrix-first-row } { . }
1346   \fi:
1347   \else:
1348   \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row:
1349   \fi:
1350 }
```

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.

```
1351 \cs_new_protected:Npn \@@_tuning_last_row:
1352 {
1353   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1354   \l_@@_code_for_last_row_tl
1355   \@@_tuning_key_small:
1356   \xglobal \colorlet { nicematrix-last-row } { . }
1357   \fi:
1358 }
```

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

```
1359 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```

1360 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1361 {
1362     \m@th
1363     $ % $

```

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

```

1364     \@@_tuning_key_small:
1365 }
1366 \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.

```

1367 \cs_new_protected:Npn \@@_begin_of_row:
1368 {
1369     \int_gincr:N \c@iRow
1370     \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1371     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1372     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1373     \pgfpicture
1374     \pgfrememberpicturepositiononpagetrue
1375     \pgfcoordinate
1376     { \@@_env: - row - \int_use:N \c@iRow - base }
1377     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1378     \str_if_empty:NF \l_@@_name_str
1379     {
1380         \pgfnodealias
1381         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1382         { \@@_env: - row - \int_use:N \c@iRow - base }
1383     }
1384     \endpgfpicture
1385 }

```

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 the L3 programming layer.

```

\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.

```

1386 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1387 {
1388   \if_int_compare:w \c@iRow = \c_zero_int
1389     \if_dim:w \box_dp:N \l_@@_cell_box > \g_@@_dp_row_zero_dim
1390       \global \g_@@_dp_row_zero_dim = \box_dp:N \l_@@_cell_box
1391     \fi:
1392     \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_zero_dim
1393       \global \g_@@_ht_row_zero_dim = \box_ht:N \l_@@_cell_box
1394     \fi:
1395   \else:
1396     \if_int_compare:w \c@iRow = \c_one_int
1397       \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_one_dim
1398         \global \g_@@_ht_row_one_dim = \box_ht:N \l_@@_cell_box
1399       \fi:
1400     \fi:
1401   \fi:
1402 }

1403 \cs_new_protected:Npn \@@_rotate_cell_box:
1404 {
1405   \box_rotate:Nn \l_@@_cell_box
1406   { \bool_if:NTF \g_@@_rotate_minus_bool { -90 } { 90 } }
1407   \bool_if:NTF \g_@@_rotate_c_bool
1408   {
1409     \hbox_set:Nn \l_@@_cell_box
1410     {
1411       \IfFormatAtLeastTF { 2026-04-01 }
1412       { \vbox_center:n { \box_use:N \l_@@_cell_box } }
1413       {
1414         \m@th
1415         $ % $
1416         \vcenter { \box_use:N \l_@@_cell_box }
1417         $ % $
1418       }
1419     }
1420   }
1421   {
1422     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1423     {
1424       \vbox_set_top:Nn \l_@@_cell_box
1425       {
1426         \vbox_to_zero:n { }
1427         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1428         \box_use:N \l_@@_cell_box
1429       }
1430     }
1431   }
1432   \bool_gset_false:N \g_@@_rotate_bool
1433   \bool_gset_false:N \g_@@_rotate_c_bool
1434   \bool_gset_false:N \g_@@_rotate_minus_bool
1435 }

```

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.

```

1436 \cs_set_protected:Npn \@@_adjust_size_box:
1437 {
1438     \if_dim:w \g_@@_blocks_wd_dim > \c_zero_dim
1439     \if_dim:w \g_@@_blocks_wd_dim > \box_wd:N \l_@@_cell_box
1440     \box_wd:N \l_@@_cell_box = \g_@@_blocks_wd_dim
1441     \fi:
1442     \global \g_@@_blocks_wd_dim = \c_zero_dim
1443     \fi:
1444     \if_dim:w \g_@@_blocks_dp_dim > \c_zero_dim
1445     \if_dim:w \g_@@_blocks_dp_dim > \box_dp:N \l_@@_cell_box
1446     \box_dp:N \l_@@_cell_box = \g_@@_blocks_dp_dim
1447     \fi
1448     \global \g_@@_blocks_dp_dim = \c_zero_dim
1449     \fi:
1450     \if_dim:w \g_@@_blocks_ht_dim > \c_zero_dim
1451     \if_dim:w \g_@@_blocks_ht_dim > \box_ht:N \l_@@_cell_box
1452     \box_ht:N \l_@@_cell_box = \g_@@_blocks_ht_dim
1453     \fi:
1454     \global \g_@@_blocks_ht_dim = \c_zero_dim
1455     \fi:
1456 }
1457 \cs_new_protected:Npn \@@_cell_end:
1458 {

```

The following command is nullified in the tabulars.

```

1459 \@@_tuning_not_tabular_end:
1460 \hbox_set_end:
1461 \@@_cell_end_i:
1462 }

```

```

\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:nNt { \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.

```

1463 \cs_new_protected:Npn \@@_cell_end_i:
1464 {

```

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.

```

1465     \g_@@_cell_after_hook_tl
1466     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1467     \@@_adjust_size_box:
1468     \box_set_ht:Nn \l_@@_cell_box
1469     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1470     \box_set_dp:Nn \l_@@_cell_box
1471     { \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”).

```

1472     \@@_update_max_cell_width:

```

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

```

1473     \@@_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.

```

1474     \bool_if:NTF \g_@@_empty_cell_bool
1475     { \box_use_drop:N \l_@@_cell_box }
1476     {
1477         \bool_if:NTF \g_@@_not_empty_cell_bool
1478         \@@_print_node_cell:
1479         {
1480             \if_dim:w \box_wd:N \l_@@_cell_box > \c_zero_dim
1481             \@@_print_node_cell:
1482             \else:
1483                 \box_use_drop:N \l_@@_cell_box
1484             \fi:

```

```

1485     }
1486   }
1487   \if_int_compare:w \c@jCol > \g_@@_col_total_int
1488     \global \g_@@_col_total_int = \c@jCol
1489   \fi:
1490   \global \let \g_@@_empty_cell_bool \c_false_bool
1491   \global \let \g_@@_not_empty_cell_bool \c_false_bool
1492 }

```

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:

```

1493 \cs_new_protected:Npn \@@_update_max_cell_width:
1494 {
1495   \if_dim:w \box_wd:N \l_@@_cell_box > \g_@@_max_cell_width_dim
1496     \global \g_@@_max_cell_width_dim = \box_wd:N \l_@@_cell_box
1497   \fi:
1498 }

```

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`).

```

1499 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1500 {
1501   \@@_math_toggle:
1502   \hbox_set_end:
1503   \bool_if:NF \g_@@_rotate_bool
1504     {
1505       \hbox_set:Nn \l_@@_cell_box
1506       {
1507         \makebox [ \l_@@_col_width_dim ] [ s ]
1508         { \hbox_unpack_drop:N \l_@@_cell_box }
1509       }
1510     }
1511   \@@_cell_end_i:
1512 }

```

```

1513 \pgfset
1514 {
1515   nicematrix / cell-node /.style =
1516   {
1517     inner~sep = \c_zero_dim ,
1518     minimum~width = \c_zero_dim
1519   }
1520 }

```

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.

```

1521 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1522 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1523 {
1524   \use:c
1525   {

```

```

1526     __siunitx_table_align_
1527     \bool_if:NTF \l__siunitx_table_text_bool
1528     \l__siunitx_table_align_text_tl
1529     \l__siunitx_table_align_number_str
1530     :n
1531   }
1532   { #1 }
1533 }

```

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!

```

1534 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1535 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1536 {
1537   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1538   \hbox:n
1539   {
1540     \pgfsys@markposition
1541     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1542   }
1543   #1
1544   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1545   \hbox:n
1546   {
1547     \pgfsys@markposition
1548     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1549   }
1550 }

1551 \cs_new_protected:Npn \@@_print_node_cell:
1552 {
1553   \socket_use:nn { nicematrix / siunitx-wrap }
1554   { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1555 }

```

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

```

1556 \cs_new_protected:Npn \@@_node_cell_active:
1557 {
1558   \pgfpicture
1559   \pgfsetbaseline \c_zero_dim
1560   \pgfrememberpicturepositiononpagetrue
1561   \pgfset { nicematrix / cell-node }
1562   \pgfnode
1563   { rectangle }
1564   { base }
1565   {

```

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).

```

1566   \sys_if_engine_xetex:T { \set@color }
1567   \box_use:N \l_@@_cell_box
1568 }
1569 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1570 { \l_@@_pgf_node_code_tl }
1571 \str_if_empty:NF \l_@@_name_str
1572 {
1573   \pgfnodealias
1574   { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1575   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }

```

```

1576     }
1577     \endpgfpicture
1578   }
1579   \cs_set_eq:NN \@@_node_cell: \@@_node_cell_active:
1580   \cs_new_protected:Npn \@@_node_cell_inactive:
1581   { \set@color \box_use_drop:N \l_@@_cell_box }

```

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).

```

1582 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1583 {
1584   \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce
1585   { g_@@_ #2 _ lines _ tl }
1586   {
1587     \use:c { @@ _ draw _ #2 : nnn }
1588     { \int_use:N \c@iRow }
1589     { \int_use:N \c@jCol }
1590     { \exp_not:n { #3 } }
1591   }
1592 }

1593 \cs_new_protected:Npn \@@_array:n
1594 {
1595   \dim_set:Nn \col@sep
1596   { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1597   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1598   { \def \@halignto { } }
1599   { \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`.

```

1600   \@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. Remark that `\str_if_eq:eeTF` is fully expandable and we need something fully expandable here. `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```

1601   [ \str_if_eq:eeTF c \l_@@_baseline_tl c t ]
1602   }
1603 \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`.

```

1604 \cs_new_eq:cN { @@_old_ar@ialign: } \ar@ialign

```

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

```

1605 \cs_new_protected:Npn \@@_create_row_node:
1606 {
1607   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1608   {
1609     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1610     \@@_create_row_node_i:
1611   }
1612 }
1613 \cs_new_protected:Npn \@@_create_row_node_i:
1614 {

```

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

```

1615   \hbox
1616   {
1617     \bool_if:NT \l_@@_code_before_bool
1618     {
1619       \vtop
1620       {
1621         \skip_vertical:N 0.5\arrayrulewidth
1622         \pgfsys@markposition
1623         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1624         \skip_vertical:N -0.5\arrayrulewidth
1625       }
1626     }
1627     \pgfpicture
1628     \pgfrememberpicturepositiononpagetrue
1629     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1630     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1631     \str_if_empty:NF \l_@@_name_str
1632     {
1633       \pgfnodealias
1634       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1635       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1636     }
1637     \endpgfpicture
1638   }
1639 }

```

```

1640 \cs_new_protected:Npn \@@_in_everycr:
1641 {
1642   \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1643   \tbl_update_cell_data_for_next_row:
1644   \int_gzero:N \c@jCol
1645   \bool_gset_false:N \g_@@_after_col_zero_bool
1646   \bool_if:NF \g_@@_row_of_col_done_bool
1647   {
1648     \@@_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).

```

1649   \clist_if_empty:NF \l_@@_hlines_clist
1650   {
1651     \str_if_eq:eeF \l_@@_hlines_clist { all }
1652     {
1653       \clist_if_in:NiT
1654       \l_@@_hlines_clist
1655       { \int_eval:n { \c@iRow + 1 } }
1656     }
1657   }

```

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.

```

1658         \int_compare:nNnT \c@iRow > { -1 }
1659         {
1660             \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1661             { \hrule height \arrayrulewidth width \c_zero_dim }
1662         }
1663     }
1664 }
1665 }
1666 }

```

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

```

1667 \cs_set_protected:Npn \@@_renew_dots:
1668 {
1669     \cs_set_eq:NN \ldots \@@_Ldots:
1670     \cs_set_eq:NN \cdots \@@_Cdots:
1671     \cs_set_eq:NN \vdots \@@_Vdots:
1672     \cs_set_eq:NN \ddots \@@_Ddots:
1673     \cs_set_eq:NN \iddots \@@_Iddots:
1674     \cs_set_eq:NN \dots \@@_Ldots:
1675     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1676 }

```

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

```

1677 \AtBeginDocument
1678 {
1679     \IfPackageLoadedTF { booktabs }
1680     {
1681         \cs_new_protected:Npn \@@_patch_booktabs:
1682         { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1683     }
1684     { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1685 }

```

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`.

```

1686 \cs_new_protected:Npn \@@_some_initialization:
1687 {
1688     \@@_everycr:
1689     \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1690     \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1691     \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1692     \dim_gzero:N \g_@@_dp_ante_last_row_dim
1693     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1694     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1695 }

```

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}`.

`\@@_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`.

```

1696 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1697 {
1698     % \tag_if_active:T % added on July 7 2026
1699     % {
1700     %     \AddToHookWithArguments{tbl/cell/init}
1701     %     {
1702     %         \int_show:n { ##2 }
1703     %         \int_show:N \g_@@_static_num_of_col_int
1704     %         \int_compare:nNnT { ##2 } > \g_@@_static_num_of_col_int
1705     %         { \SuspendTagging }
1706     %     }
1707     % }

```

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

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

```

1708     \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

1709     \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1710     \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.

```

1711     \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.

```

1712     \fp_gzero:N \g_@@_total_X_weight_fp
1713     \bool_gset_false:N \g_@@_V_of_X_bool
1714     \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1715     \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol
1716     \@@_patch_booktabs:
1717     \box_clear_new:N \l_@@_cell_box
1718     \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}`).

```

1719     \bool_if:NT \l_@@_small_bool
1720     {
1721         \def \arraystretch { 0.47 }
1722         \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1723     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1724 }

```

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!

```

1725   \bool_if:NT \g_@@_create_cell_nodes_bool
1726   {
1727     \tbl_put_right:Nn \@@_begin_of_row:
1728     {
1729       \pgfsys@markposition
1730       { \@@_env: - row - \int_use:N \c@iRow - base }
1731     }
1732     \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1733   }

```

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`.

```

1734   \def \ar@ialign
1735   {
1736     \tbl_init_cell_data_for_table:
1737     \@@_some_initialization:
1738     \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`.

```

1739     \cs_set_eq:Nc \ar@ialign { \@@_old_ar@ialign: }
1740     \halign
1741   }

```

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).

```

1742   \cs_set_eq:NN \@@_old_ldots: \ldots
1743   \cs_set_eq:NN \@@_old_cdots: \cdots
1744   \cs_set_eq:NN \@@_old_vdots: \vdots
1745   \cs_set_eq:NN \@@_old_ddots: \ddots
1746   \cs_set_eq:NN \@@_old_iddots: \iddots
1747   \bool_if:NTF \l_@@_standard_cline_bool
1748   { \cs_set_eq:NN \cline \@@_standard_cline: }
1749   { \cs_set_eq:NN \cline \@@_cline: }
1750   \cs_set_eq:NN \Ldots \@@_Ldots:
1751   \cs_set_eq:NN \Cdots \@@_Cdots:
1752   \cs_set_eq:NN \Vdots \@@_Vdots:
1753   \cs_set_eq:NN \Ddots \@@_Ddots:
1754   \cs_set_eq:NN \Iddots \@@_Iddots:
1755   \cs_set_eq:NN \Hline \@@_Hline:
1756   \cs_set_eq:NN \Hspace \@@_Hspace:
1757   \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1758   \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1759   \cs_set_eq:NN \Block \@@_Block:
1760   \cs_set_eq:NN \rotate \@@_rotate:
1761   \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1762   \cs_set_eq:NN \dotfill \@@_dotfill:
1763   \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1764   \cs_if_free:NT \Body { \cs_set_eq:NN \Body \@@_Body: }
1765   \cs_if_free:NT \CodeBefore { \cs_set_eq:NN \CodeBefore \@@_CodeBefore: }
1766   \cs_set_eq:NN \diagbox \@@_diagbox:nn
1767   \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1768   \cs_set_eq:NN \TopRule \@@_TopRule
1769   \cs_set_eq:NN \MidRule \@@_MidRule
1770   \cs_set_eq:NN \BottomRule \@@_BottomRule
1771   \cs_set_eq:NN \RowStyle \@@_RowStyle:n

```

```

1772 \cs_set_eq:NN \Hbrace \@@_Hbrace
1773 \cs_set_eq:NN \Vbrace \@@_Vbrace
1774 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1775 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1776 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1777 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1778 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1779 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1780 \cs_set_eq:NN \FalseRow \@@_FalseRow
1781 \int_if_zero:nTF \l_@@_first_row_int
1782 { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_first_last_row: }
1783 {
1784   \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1785   { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row: }
1786 }
1787 \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:n` will be put in `\@@_after_array:`.

```

1788 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1789 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1790 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1791 \@@_revert_colortbl:

```

If there is one or several commands `\tablarnote` in the caption specified by the key `caption` and if that caption has to be composed above the tabular, we have now that information because it has been written in the `aux` file at a previous run. We use that information to start counting the tabular notes in the main array at the right value (we remember that the caption will be composed *after* the array!).

```

1792 \tl_if_exist:NT \l_@@_note_in_caption_tl
1793 {
1794   \tl_if_empty:NF \l_@@_note_in_caption_tl
1795   {
1796     \int_gset:Nn \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1797     \int_gset:Nn \c@tablarnote \l_@@_note_in_caption_tl
1798   }
1799 }

```

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).

```

1800 \seq_gclear:N \g_@@_multicolumn_cells_seq
1801 \seq_gclear:N \g_@@_multicolumn_sizes_seq

```

When we compose a cell which belongs to a column which contains a instruction `\columncolor` (in the preamble of the environment), we add the number of that column in the following sequence (in order to recall that we have written the following instruction of the `\g_@@_pre_code_before_tl`).

```

1802 \seq_gclear_new:N \g_@@_columncolor_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).

```

1803 \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`).

```

1804 \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.

```

1805 \int_gzero:N \g_@@_col_total_int

```

```

1806 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1807 \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.

```

1808 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1809 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1810 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1811 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1812 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1813 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1814 \tl_gclear:N \g_nicematrix_code_before_tl
1815 \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]`).

```

1816 \dim_zero_new:N \l_@@_left_delim_dim
1817 \dim_zero_new:N \l_@@_right_delim_dim
1818 \bool_if:NTF \g_@@_delims_bool
1819 {

```

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

```

1820 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1821 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1822 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1823 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1824 }
1825 {
1826 \dim_gset:Nn \l_@@_left_delim_dim
1827 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1828 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1829 }
1830 }

```

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.

```

1831 \cs_new_protected:Npn \@@_pre_array:
1832 {
1833 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1834 \int_gzero_new:N \c@iRow
1835 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1836 \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).

```

1837 \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1838 { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1839 \int_compare:nNnT \l_@@_last_col_int > \c_zero_int
1840 { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1841 \bool_if:NT \g_@@_aux_found_bool
1842 {
1843 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1844 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1845 \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1846 \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1847 }

```

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).

```

1848 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1849 {
1850   \bool_set_true:N \l_@@_last_row_without_value_bool
1851   \bool_if:NT \g_@@_aux_found_bool
1852     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1853 }
1854 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1855 {
1856   \bool_if:NT \g_@@_aux_found_bool
1857     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1858 }

```

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”.

```

1859 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1860 {
1861   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1862     {
1863       \dim_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1864         { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1865       \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1866         { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1867     }
1868 }

1869 \seq_gclear:N \g_@@_cols_vlism_seq
1870 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1871 \bool_if:NT \l_@@_code_before_bool \@@_exec_code_before:

```

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

```

1872 \@@_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).

```

1873 \hbox_set:Nw \l_@@_the_array_box
1874 \skip_horizontal:N \l_@@_left_margin_dim % \skip_horizontal:N ?
1875 \skip_horizontal:N \l_@@_extra_left_margin_dim
1876 \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).

```

1877 \m@th
1878 $ % $
1879 \bool_if:NTF \l_@@_light_syntax_bool
1880   { \use:c { @@-light-syntax } }
1881   { \use:c { @@-normal-syntax } }
1882 }

```

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

```

1883 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1884 {
1885   \tl_set:Nn \l_tmpa_tl { #1 }
1886   \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1887     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1888   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1889   \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...

```

1890   \@@_pre_array:
1891 }

```

9 The `\CodeBefore`

```

1892 \cs_new_protected_nopar:Npn \@@_Body: { \@@_fatal:n { Body~alone } }
1893 \cs_new_protected_nopar:Npn \@@_CodeBefore: { \@@_fatal:n { Misuse-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).

```

1894 \cs_new_protected:Npn \@@_pre_code_before:
1895 {

```

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.

```

1896   \pgfsys@markposition { \@@_env: - position }
1897   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1898   \pgfpicture
1899   \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1900   \int_step_inline:nnm \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1901   {
1902     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1903     \pgfcoordinate { \@@_env: - row - ##1 }
1904       { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1905   }

```

Now, the recreation of the `col` nodes.

```

1906   \int_step_inline:nnm \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1907   {
1908     \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1909     \pgfcoordinate { \@@_env: - col - ##1 }
1910       { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1911   }

```

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

```

1912   \bool_if:NT \g_@@_create_cell_nodes_bool \@@_recreate_cell_nodes:
1913   \endpgfpicture

```

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

```

1914   \@@_create_diag_nodes:

```

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

```

1915   \@@_create_blocks_nodes:
1916   \IfPackageLoadedT { tikz }
1917   {
1918     \tikzset

```

```

1919     {
1920         every-picture / .style =
1921             { overlay , name~prefix = \@@_env: - }
1922     }
1923 }
1924 \cs_set_eq:NN \cellcolor \@@_cellcolor
1925 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1926 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1927 \cs_set_eq:NN \rowcolor \@@_rowcolor
1928 \cs_set_eq:NN \rowcolors \@@_rowcolors
1929 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1930 \cs_set_eq:NN \arraycolor \@@_arraycolor
1931 \cs_set_eq:NN \columncolor \@@_columncolor
1932 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1933 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1934 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1935 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNamesCodeBefore
1936 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1937 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1938 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1939 }

1940 \cs_new_protected:Npn \@@_exec_code_before:
1941 {

```

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 detect whether we actually have coloring instructions to execute...

```

1942 \clist_map_inline:Nn \l_@@_corners_cells_clist
1943 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1944 \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`.

```

1945 \@@_add_to_colors_seq:nn { { nocolor } } { }
1946 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1947 \group_begin:
1948 \@@_security_font_commands:

```

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

```

1949 \if_mode_math:
1950 \@@_exec_code_before_i:
1951 \else:
1952 $ % $
1953 \@@_exec_code_before_i:
1954 $ % $
1955 \fi:
1956 \group_end:
1957 }

```

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`).

```

1958 \cs_new_protected:Npn \@@_exec_code_before_i:
1959 {
1960 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1961 { \@@_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`.

```

1962 \exp_last_unbraced:No \@@_CodeBefore_keys:
1963 \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.

```

1964 \@@_actually_color:
1965 \l_@@_code_before_tl
1966 \q_stop
1967 }

```

```

1968 \keys_define:nn { nicematrix / CodeBefore }
1969 {
1970   create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1971   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1972   sub-matrix .value_required:n = true ,
1973   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1974   delimiters / color .value_required:n = true ,
1975   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1976 }

1977 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1978 {
1979   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1980   \@@_CodeBefore:w
1981 }

```

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.

```

1982 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1983 {
1984   \bool_if:NTF \g_@@_aux_found_bool
1985   {
1986     \@@_pre_code_before:
1987     \legacy_if:nF { measuring@ } { #1 }
1988   }

```

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).

```

1989 {
1990   \pgfsys@markposition { \@@_env: - position }
1991   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1992   \pgfpicture
1993   \pgf@relevantforpicturesizefalse
1994   \endpgfpicture

```

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

```

1995 \pgfpicture
1996 \pgfrememberpicturepositiononpagetrue
1997 \endpgfpicture

```

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

```

1998 \pgfpicture
1999 \pgf@relevantforpicturesizefalse
2000 \pgfrememberpicturepositiononpagetrue
2001 \endpgfpicture

```

The following picture corresponds `\@@_actually_color:`

```

2002 \pgfpicture
2003 \pgf@relevantforpicturesizefalse
2004 \endpgfpicture
2005 }
2006 }

```


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.

```

2007 \cs_new_protected:Npn \@@_recreate_cell_nodes:
2008 {
2009   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
2010   {
2011     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
2012     \pgfcoordinate { \@@_env: - row - ##1 - base }
2013     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
2014     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
2015     {
2016       \cs_if_exist:cT
2017       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
2018       {
2019         \pgfsys@getposition
2020         { \@@_env: - ##1 - #####1 - NW }
2021         \@@_node_position:
2022         \pgfsys@getposition
2023         { \@@_env: - ##1 - #####1 - SE }
2024         \@@_node_position_i:
2025         \@@_pgf_rect_node:nnn
2026         { \@@_env: - ##1 - #####1 }
2027         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
2028         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
2029       }
2030     }
2031   }
2032   \@@_create_extra_nodes:
2033   \@@_create_aliases_last:
2034 }

2035 \cs_new_protected:Npn \@@_create_aliases_last:
2036 {
2037   \int_step_inline:nn \c@iRow
2038   {
2039     \pgfnodealias
2040     { \@@_env: - ##1 - last }
2041     { \@@_env: - ##1 - \int_use:N \c@jCol }
2042   }
2043   \int_step_inline:nn \c@jCol
2044   {
2045     \pgfnodealias
2046     { \@@_env: - last - ##1 }
2047     { \@@_env: - \int_use:N \c@iRow - ##1 }
2048   }
2049   \pgfnodealias
2050   { \@@_env: - last - last }
2051   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
2052 }

2053 \cs_new_protected:Npn \@@_create_blocks_nodes:
2054 {
2055   \pgfpicture
2056   \pgf@relevantforpicturesizefalse
2057   \pgfrememberpicturepositiononpagetrue
2058   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
2059   { \@@_create_one_block_node:nnnnn ##1 }
2060   \endpgfpicture
2061 }

```

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

```

2062 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
2063 {
2064   \tl_if_empty:nF { #5 }
2065   {
2066     \@@_qpoint:n { col - #2 }
2067     \dim_set_eq:NN \l_tmpa_dim \pgf@x
2068     \@@_qpoint:n { #1 }
2069     \dim_set_eq:NN \l_tmpb_dim \pgf@y
2070     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
2071     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
2072     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
2073     \@@_pgf_rect_node:nnnnn
2074       { \@@_env: - #5 }
2075       { \dim_use:N \l_tmpa_dim }
2076       { \dim_use:N \l_tmpb_dim }
2077       { \dim_use:N \l_@@_tmpc_dim }
2078       { \dim_use:N \pgf@y }
2079   }
2080 }

```

10 The environment `{NiceArrayWithDelims}`

```

2081 \NewDocumentEnvironment { NiceArrayWithDelims }
2082 { m m O { } m ! O { } t \CodeBefore }
2083 {
2084   \@@_provide_pgfsyspdfmark:
2085   \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.

```

2086   \bgroup

2087   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2088   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2089   \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
2090   \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

2091   \int_gzero:N \g_@@_block_box_int
2092   \dim_gzero:N \g_@@_width_last_col_dim
2093   \dim_gzero:N \g_@@_width_first_col_dim
2094   \bool_gset_false:N \g_@@_row_of_col_done_bool
2095   \str_if_empty:NT \g_@@_name_env_str
2096     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
2097   \bool_if:NTF \l_@@_tabular_bool
2098     \mode_leave_vertical:
2099     \@@_test_if_math_mode:
2100   \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
2101   \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.

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).

⁸e.g. `\color[rgb]{0.5,0.5,0}`

```
2102 \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.

```
2103 \cs_if_exist:NT \tikz@library@external@loaded
2104 {
2105     \tikzexternaldisable
2106     \cs_if_exist:NT \ifstandalone
2107     { \tikzset { external / optimize = false } }
2108 }
```

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

```
2109 \int_gincr:N \g_@@_env_int
2110 \bool_if:NF \l_@@_block_auto_columns_width_bool
2111 { \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.

```
2112 \seq_gclear:N \g_@@_blocks_seq
2113 \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`.

```
2114 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
2115 \seq_gclear:N \g_@@_pos_of_xdots_seq
2116 \tl_gclear_new:N \g_@@_code_before_tl
2117 \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.

```
2118 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2119 {
2120     \bool_gset_true:N \g_@@_aux_found_bool
2121     \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2122 }
2123 { \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.

```
2124 \tl_gclear:N \g_@@_aux_tl
2125 \tl_if_empty:NF \g_@@_code_before_tl
2126 {
2127     \bool_set_true:N \l_@@_code_before_bool
2128     \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
2129 }
2130 \tl_if_empty:NF \g_@@_pre_code_before_tl
2131 { \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`.

```
2132 \bool_if:NTF \g_@@_delims_bool
2133 { \keys_set:nn { nicematrix / pNiceArray } }
2134 { \keys_set:nn { nicematrix / NiceArray } }
2135 { #3 , #5 }
```

```
2136 \@@_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:.`

```

2137 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
2138 }

```

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

```

2139 {
2140   \bool_if:nTF \l_@@_light_syntax_bool
2141     { \use:c { end @@-light-syntax } }
2142     { \use:c { end @@-normal-syntax } }
2143   $ % $
2144   \skip_horizontal:N \l_@@_right_margin_dim
2145   \skip_horizontal:N \l_@@_extra_right_margin_dim
2146   \hbox_set_end:
2147   \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.

```

2148 \bool_if:NT \l_@@_width_used_bool
2149 {
2150   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2151     { \@@_error_or_warning:n { width-without-X-columns } }
2152 }

```

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

```

2153 \fp_compare:nNnT \g_@@_total_X_weight_fp > \c_zero_fp
2154   \@@_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).

```

2155 \int_compare:nNnT \l_@@_last_row_int > { -2 }
2156 {
2157   \bool_if:NF \l_@@_last_row_without_value_bool
2158   {
2159     \int_compare:nNnF \l_@@_last_row_int = \c_iRow
2160     {
2161       \@@_error:n { Wrong-last-row }
2162       \int_set_eq:NN \l_@@_last_row_int \c_iRow
2163     }
2164   }
2165 }

```

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

```

2166 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2167 \bool_if:nTF \g_@@_last_col_found_bool
2168   { \int_gdecr:N \c@jCol }
2169   {
2170     \int_compare:nNnT \l_@@_last_col_int > { -1 }
2171     { \@@_error:n { last-col-not-used } }
2172   }

```

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

```

2173 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2174 \int_compare:nNnT \l_@@_last_row_int > { -1 }
2175   { \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. 96).

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

```

2176 \int_if_zero:nT \l_@@_first_col_int
2177 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2178 \bool_if:nTF { ! \g_@@_delims_bool }
2179 {
2180   \str_if_eq:eeTF c \l_@@_baseline_tl
2181   \@@_use_arraybox_with_notes_c:
2182   {
2183     \str_if_eq:eeTF b \l_@@_baseline_tl
2184     \@@_use_arraybox_with_notes_b:
2185     \@@_use_arraybox_with_notes:
2186   }
2187 }

```

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).

```

2188 {
2189   \int_if_zero:nTF \l_@@_first_row_int
2190   {
2191     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2192     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2193   }
2194   { \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

```

2195   \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2196   {
2197     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2198     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2199   }
2200   { \dim_zero:N \l_tmpb_dim }
2201 \hbox_set:Nn \l_tmpa_box
2202 {
2203   \m@th
2204   $ % $
2205   \@@_color:o \l_@@_delimiters_color_tl
2206   \exp_after:wN \left \g_@@_left_delim_tl
2207   \vcenter
2208   {

```

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.

```

2209     \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2210     \hbox
2211     {
2212       \bool_if:NTF \l_@@_tabular_bool
2213       { \skip_horizontal:n { - \tabcolsep } }
2214       { \skip_horizontal:n { - \arraycolsep } }
2215       \@@_use_arraybox_with_notes_c:
2216       \bool_if:NTF \l_@@_tabular_bool
2217       { \skip_horizontal:n { - \tabcolsep } }
2218       { \skip_horizontal:n { - \arraycolsep } }
2219     }

```

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

```

2220     \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2221   }
2222   \exp_after:wN \right \g_@@_right_delim_tl
2223   $ % $
2224 }

```

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

Now, the box `\l_tmpa_box` is created with the correct delimiters.

We will put the box in the TeX flow. However, we have a small work to do when the option `delimiters/max-width` is used.

```

2225     \bool_if:NTF \l_@@_delimiters_max_width_bool
2226     { \@@_put_box_in_flow_bis:nn \g_@@_left_delim_tl \g_@@_right_delim_tl }
2227     \@@_put_box_in_flow:
2228 }

```

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. 97).

```

2229     \bool_if:NT \g_@@_last_col_found_bool
2230     { \skip_horizontal:N \g_@@_width_last_col_dim }
2231     \bool_if:NT \l_@@_preamble_bool
2232     {
2233         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2234         { \@@_err_columns_not_used: }
2235     }
2236     \@@_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.

```

2237     \egroup

```

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

```

2238     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2239     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2240     \iow_now:Ne \@mainaux
2241     {
2242         \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2243         \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2244         { \exp_not:o \g_@@_aux_tl }
2245     }
2246     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2247     \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2248 }

```

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

```

2249 \cs_new_protected:Npn \@@_err_columns_not_used:
2250 {
2251     \@@_warning:n { columns-not-used }
2252     \cs_gset:Npn \@@_err_columns_not_used: { }
2253 }

```

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 .

```

2254 \cs_new_protected:Npn \@@_compute_width_X:
2255 {
2256     \tl_gput_right:Ne \g_@@_aux_tl
2257     {
2258         \bool_set_true:N \l_@@_X_columns_aux_bool
2259         \dim_set:Nn \l_@@_X_columns_dim
2260         {

```

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`

```

2261         \bool_lazy_and:nnTF \g_@@_V_of_X_bool \l_@@_X_columns_aux_bool
2262         { \dim_use:N \l_@@_X_columns_dim }
2263         {
2264             \dim_compare:nNnTF

```

```

2265     {
2266         \dim_abs:n
2267         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2268     }
2269     <
2270     { 0.001 pt }
2271     { \dim_use:N \l_@@_X_columns_dim }
2272     {
2273         \dim_eval:n
2274         {
2275             \l_@@_X_columns_dim
2276             +
2277             \fp_to_dim:n
2278             {
2279                 (
2280                     \dim_eval:n
2281                     { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2282                 )
2283                 / \fp_use:N \g_@@_total_X_weight_fp
2284             }
2285         }
2286     }
2287 }
2288 }
2289 }
2290 }

```

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`.

```

2291 \cs_new_protected:Npn \@@_transform_preamble:
2292 {
2293     \@@_transform_preamble_i:
2294     \@@_transform_preamble_ii:
2295 }
2296 \cs_new_protected:Npn \@@_transform_preamble_i:
2297 {
2298     \int_gzero:N \c@jCol

```

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

```

2299     \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.

```

2300     \bool_gset_false:N \g_tmpb_bool

```

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

```

2301     \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2302     \int_zero:N \l_tmpa_int
2303     \tl_gclear:N \g_@@_array_preamble_tl
2304     \str_if_eq:eeTF \l_@@_vlines_clist { all }
2305     {
2306         \tl_gset:Nn \g_@@_array_preamble_tl

```

```

2307         { ! { \skip_horizontal:N \arrayrulewidth } }
2308     }
2309     {
2310         \clist_if_in:NnT \l_@@_vlines_clist 1
2311         {
2312             \tl_gset:Nn \g_@@_array_preamble_tl
2313                 { ! { \skip_horizontal:N \arrayrulewidth } }
2314         }
2315     }

```

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

```

2316     \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2317     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2318     \@@_replace_columncolor:
2319 }

2320 \cs_new_protected:Npn \@@_transform_preamble_ii:
2321 {

```

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

```

2322     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2323     {
2324         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2325         { \bool_gset_true:N \g_@@_delims_bool }
2326     }
2327     { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2328     \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).

```

2329     \int_if_zero:nTF \l_@@_first_col_int
2330     { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2331     {
2332         \bool_if:NF \g_@@_delims_bool
2333         {
2334             \bool_if:NF \l_@@_tabular_bool
2335             {
2336                 \clist_if_empty:NT \l_@@_vlines_clist
2337                 {
2338                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2339                     { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2340                 }
2341             }
2342         }
2343     }
2344     \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2345     { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2346     {
2347         \bool_if:NF \g_@@_delims_bool
2348         {
2349             \bool_if:NF \l_@@_tabular_bool
2350             {
2351                 \clist_if_empty:NT \l_@@_vlines_clist
2352                 {
2353                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2354                     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2355                 }

```



```

2356     }
2357   }
2358 }

```

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.

```

2359   \tag_if_active:F
2360   {

```

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

```

2361     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2362     {
2363       \tl_gput_right:Nn \g_@@_array_preamble_tl
2364       { > { \@@_err_too_many_cols: } 1 }
2365     }
2366   }
2367 }

```

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.

```

2368 \cs_new_protected:Npn \@@_rec_preamble:n #1
2369 {

```

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

```

2370   \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2371   { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2372   {

```

Now, the columns defined by \newcolumntype of array.

```

2373     \cs_if_exist:cTF { NC @ find @ #1 }
2374     {
2375       \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2376       \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2377     }
2378     {
2379       \str_if_eq:nnTF { #1 } { S }
2380       { \@@_fatal:n { unknown~column~type~S } }
2381       { \@@_fatal:nn { unknown~column~type } { #1 } }
2382     }
2383   }
2384 }

```

For c, l and r

```

2385 \cs_new_protected:Npn \@@_c: #1
2386 {
2387   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2388   \tl_gclear:N \g_@@_pre_cell_tl
2389   \tl_gput_right:Nn \g_@@_array_preamble_tl
2390   { > \@@_cell_begin: c < \@@_cell_end: }

```

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.

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

```

2391 \int_gincr:N \c@jCol
2392 \@@_rec_preamble_after_col:n
2393 }

2394 \cs_new_protected:Npn \@@_l: #1
2395 {
2396 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2397 \tl_gclear:N \g_@@_pre_cell_tl
2398 \tl_gput_right:Nn \g_@@_array_preamble_tl
2399 {
2400 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2401 l
2402 < \@@_cell_end:
2403 }
2404 \int_gincr:N \c@jCol
2405 \@@_rec_preamble_after_col:n
2406 }

2407 \cs_new_protected:Npn \@@_r: #1
2408 {
2409 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2410 \tl_gclear:N \g_@@_pre_cell_tl
2411 \tl_gput_right:Nn \g_@@_array_preamble_tl
2412 {
2413 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2414 r
2415 < \@@_cell_end:
2416 }
2417 \int_gincr:N \c@jCol
2418 \@@_rec_preamble_after_col:n
2419 }

```

For ! and @

```

2420 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2421 {
2422 \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2423 \@@_rec_preamble:n
2424 }
2425 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2426 \cs_new_protected:cpn { @@ _ | : } #1
2427 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2428 \int_incr:N \l_tmpa_int
2429 \@@_make_preamble_i_i:n
2430 }

2431 \cs_new_protected:cpn { @@ _ F : } #1 % added 2026/06/28
2432 {
2433 \tl_gput_right:Nn \g_@@_array_preamble_tl
2434 {
2435 > {
2436 \skip_horizontal:n { -2 \col@sep + \l_@@_width_of_false_dim }
2437 \skip_horizontal:N \c_zero_dim
2438 \@@_cell_begin:
2439 }
2440 c
2441 < {
2442 \@@_cell_end:
2443 \columncolor { nocolor }
2444 }
2445 }

```

```

2446 \int_gincr:N \c@jCol
2447 \tl_gput_left:Ne \g_nicematrix_code_before_tl
2448 { \exp_not:N \EmptyColumn { \arabic{jCol} } }
2449 \@@_rec_preamble_after_col:n
2450 }
2451 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2452 {

```

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

```

2453 \str_if_eq:nnTF { #1 } { | }
2454 { \use:c { @@_ | : } | }
2455 { \@@_make_preamble_i_ii:nn { } #1 }
2456 }

```

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

```

2457 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2458 {
2459   \str_if_eq:nnTF { #2 } { [ ]
2460     { \@@_make_preamble_i_ii:nw { #1 } [ ]
2461       { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2462     }
2463   \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2464   { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2465   \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2466   {
2467     \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2468     \tl_gput_right:Ne \g_@@_array_preamble_tl
2469     {

```

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

```

2470 \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_before_dim }
2471 }
2472 \tl_gput_right:Ne \g_@@_rules_tl
2473 {

```

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:

```

2474 {
2475   \int_compare:nNt \l_tmpa_int > 1
2476   { \@@_set_multiplicity:n { \int_use:N \l_tmpa_int } }
2477   \int_set:Nn \l_@@_position_int { \int_eval:n { \c@jCol + 1 } }
2478   \dim_set:Nn \l_@@_rule_width_after_dim
2479   { \dim_use:N \l_@@_rule_width_before_dim }
2480   \@@_draw_vrule:n { #2 }
2481 }

```

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.

```

2482 }
2483 \int_zero:N \l_tmpa_int
2484 \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2485 \@@_rec_preamble:n #1
2486 }

```

```

2487 \cs_new_protected:cpn { @@ _ > : } #1 #2
2488 {
2489   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2490   \@@_rec_preamble:n
2491 }
2492 \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.

```

2493 \keys_define:nn { nicematrix / p-column }
2494 {
2495   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2496   r .value_forbidden:n = true ,
2497   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2498   c .value_forbidden:n = true ,
2499   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2500   l .value_forbidden:n = true ,
2501   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2502   S .value_forbidden:n = true ,
2503   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2504   p .value_forbidden:n = true ,
2505   t .meta:n = p ,
2506   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2507   m .value_forbidden:n = true ,
2508   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2509   b .value_forbidden:n = true
2510 }

```

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

```

2511 \cs_new_protected:Npn \@@_p: #1
2512 {
2513   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2514   \@@_make_preamble_ii_i:n
2515 }
2516 \cs_new_eq:NN \@@_b: \@@_p:
2517 \cs_new_eq:NN \@@_m: \@@_p:
2518 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2519 {
2520   \str_if_eq:nnTF { #1 } { [ ]
2521     { \@@_make_preamble_ii_ii:w [ ] }
2522     { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2523   }
2524 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2525 { \@@_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.

```

2526 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2527 {

```

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).

```

2528   \str_set:Nn \l_@@_hpos_col_str { j }
2529   \@@_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`.

```

2530   \setlength { \l_tmpa_dim } { #2 }

```

```

2531 \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2532 }
2533 \cs_new_protected:Npn \@@_keys_p_column:n #1
2534 { \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.

```

2535 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2536 {

```

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

```

2537 \use:e
2538 {
2539 \@@_make_preamble_ii_vi:nnnnnnnn
2540 { \str_if_eq:eeTF p \l_@@_vpos_col_str { t } { b } }
2541 { #1 }
2542 {
2543 \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.

```

2544 \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2545 { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2546 {

```

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

```

2547 \def \exp_not:N \l_@@_hpos_cell_tl
2548 { \str_lowercase:f { \l_@@_hpos_col_str } }
2549 }
2550 \IfPackageLoadedTF { ragged2e }
2551 {
2552 \str_case:on \l_@@_hpos_col_str
2553 {

```

The following `\exp_not:N` are mandatory.

```

2554 c { \exp_not:N \Centering }
2555 l { \exp_not:N \RaggedRight }
2556 r { \exp_not:N \RaggedLeft }
2557 }
2558 }
2559 {
2560 \str_case:on \l_@@_hpos_col_str
2561 {
2562 c { \exp_not:N \centering }
2563 l { \exp_not:N \raggedright }
2564 r { \exp_not:N \raggedleft }
2565 }
2566 }
2567 #3
2568 }
2569 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2570 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2571 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2572 { #2 }
2573 {
2574 \str_case:onF \l_@@_hpos_col_str
2575 {
2576 { j } { c }
2577 { si } { c }
2578 }

```

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

```

2579         { \str_lowercase:f \l_@@_hpos_col_str }
2580     }
2581 }

```

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

```

2582     \int_gincr:N \c@jCol
2583     \@@_rec_preamble_after_col:n
2584 }

```

#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*.

```

2585 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2586 {
2587     \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2588     {
2589         \tl_gput_right:Nn \g_@@_array_preamble_tl
2590         { > \@@_test_if_empty_for_S: }
2591     }
2592     {
2593         \str_if_eq:eeTF { #7 } { varwidth }
2594         {
2595             \tl_gput_right:Nn \g_@@_array_preamble_tl
2596             { > \@@_test_if_empty_varwidth: }
2597         }
2598         { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2599     }
2600     \tl_gput_right:Nn \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2601     \tl_gclear:N \g_@@_pre_cell_tl
2602     \tl_gput_right:Nn \g_@@_array_preamble_tl
2603     {
2604         > {

```

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.

```

2605         \dim_set:Nn \l_@@_col_width_dim { #2 }
2606         \@@_cell_begin:

```

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

```

2607         \use:c { #7 } [ #1 ] { #2 }

```

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

```

2608     \everypar
2609     {
2610         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2611         \everypar { }
2612     }

```

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

```

2613     #3

```

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

```

2614         \g_@@_row_style_tl
2615         \arraybackslash
2616         #5
2617     }
2618     #8
2619     < {
2620         #6

```

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

```

2621         \@finalstrut \@arstrutbox
2622         \use:c { end #7 }

```

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

```

2623         #4
2624         \@@_cell_end:
2625     }
2626 }
2627 }

```

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

```

2628 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2629 {

```

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 `&`).

```

2630     \group_align_safe_begin:
2631     \peek_meaning:NTF &
2632     \@@_the_cell_is_empty:
2633     {
2634         \peek_meaning:NTF \\\
2635         \@@_the_cell_is_empty:
2636         {
2637             \peek_meaning:NTF \crcr
2638             \@@_the_cell_is_empty:
2639             \group_align_safe_end:
2640         }
2641     }
2642 }

```

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

```

2643 \cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
2644 {
2645     \group_align_safe_begin:
2646     \peek_meaning:NTF &
2647     \@@_the_cell_is_empty_varwidth:
2648     {
2649         \peek_meaning:NTF \\\
2650         \@@_the_cell_is_empty_varwidth:
2651         {
2652             \peek_meaning:NTF \crcr
2653             \@@_the_cell_is_empty_varwidth:
2654             \group_align_safe_end:
2655         }
2656     }
2657 }
2658 \cs_new_protected:Npn \@@_the_cell_is_empty:
2659 {
2660     \group_align_safe_end:
2661     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2662     {

```

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

```
2663 \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).

```
2664 \skip_horizontal:N \l_@@_col_width_dim
```

```
2665 }
```

```
2666 }
```

```
2667 \cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
```

```
2668 {
```

```
2669 \group_align_safe_end:
```

```
2670 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
```

```
2671 { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
```

```
2672 }
```

```
2673 \cs_new_protected:Npn \@@_test_if_empty_for_S:
```

```
2674 {
```

```
2675 \peek_meaning:NT \__siunitx_table_skip:n
```

```
2676 { \bool_gset_true:N \g_@@_empty_cell_bool }
```

```
2677 }
```

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.

```
2678 \cs_new_protected:Npn \@@_center_cell_box:
```

```
2679 {
```

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

```
2680 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
```

```
2681 {
```

```
2682 \dim_compare:nNnT
```

```
2683 { \box_ht:N \l_@@_cell_box }
```

```
2684 >
```

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).

```
2685 { \box_ht:N \strutbox }
```

```
2686 {
```

```
2687 \hbox_set:Nn \l_@@_cell_box
```

```
2688 {
```

```
2689 \box_move_down:nn
```

```
2690 {
```

```
2691 ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2692 + \baselineskip ) / 2
```

```
2693 }
```

```
2694 { \box_use:N \l_@@_cell_box }
```

```
2695 }
```

```
2696 }
```

```
2697 }
```

```
2698 }
```

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

```
2699 \cs_new_protected:Npn \@@_V: #1 #2
```

```
2700 {
```

```
2701 \str_if_eq:nnTF { #2 } { [ ] }
```

```
2702 { \@@_make_preamble_V_i:w [ ] }
```

```
2703 { \@@_make_preamble_V_i:w [ ] { #2 } }
```

```
2704 }
```



```

2705 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2706 { \@@_make_preamble_V_ii:nn { #1 } }
2707 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2708 {
2709   \str_set:Nn \l_@@_vpos_col_str { p }
2710   \str_set:Nn \l_@@_hpos_col_str { j }
2711   \@@_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`.

```

2712 \setlength { \l_tmpa_dim } { #2 }
2713 \IfPackageLoadedTF { varwidth }
2714 { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2715 {
2716   \@@_error_or_warning:n { varwidth-not-loaded }
2717   \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2718 }
2719 }
2720 % \end{macrocod}
2721 %
2722 % \medskip
2723 % For |w| and |W|
2724 % \begin{macrocode}
2725 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2726 \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.

```

2727 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3
2728 {
2729   \tl_if_in:nnTF { clr } { #3 }
2730   { \@@_make_preamble_w_i:nnnn { #1 } { #2 } { #3 } }
2731   {
2732     \@@_error:nn { Invalid-argument-for-w } { #3 }
2733     \@@_make_preamble_w_i:nnnn { #1 } { #2 } { c }
2734   }
2735 }
2736 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2 #3 #4
2737 {
2738   \str_if_eq:nnTF { #3 } { s }
2739   { \@@_make_preamble_w_ii:nnnn { #1 } { #4 } }
2740   { \@@_make_preamble_w_iii:nnnn { #1 } { #2 } { #3 } { #4 } }
2741 }

```

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.

```

2742 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2
2743 {
2744   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2745   \tl_gclear:N \g_@@_pre_cell_tl
2746   \tl_gput_right:Nn \g_@@_array_preamble_tl
2747   {
2748     > {

```

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`.

```

2749         \setlength { \l_@@_col_width_dim } { #2 }
2750         \@@_cell_begin:
2751         \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2752     }
2753     c
2754     < {
2755         \@@_cell_end_for_w_s:
2756         #1
2757         \@@_adjust_size_box:
2758         \box_use_drop:N \l_@@_cell_box
2759     }
2760 }
2761 \int_gincr:N \c@jCol
2762 \@@_rec_preamble_after_col:n
2763 }

```

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

```

2764 \cs_new_protected:Npn \@@_make_preamble_w_iii:nnnn #1 #2 #3 #4
2765 {
2766     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2767     \tl_gclear:N \g_@@_pre_cell_tl
2768     \tl_gput_right:Nn \g_@@_array_preamble_tl
2769     {
2770         > {

```

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`.

```

2771         \setlength { \l_@@_col_width_dim } { #4 }
2772         \hbox_set:Nw \l_@@_cell_box
2773         \@@_cell_begin:
2774         \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2775     }
2776     c
2777     < {
2778         \@@_cell_end:
2779         \hbox_set_end:
2780         #1
2781         \@@_adjust_size_box:
2782         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2783     }
2784 }

```

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

```

2785     \int_gincr:N \c@jCol
2786     \@@_rec_preamble_after_col:n
2787 }

2788 \cs_new_protected:Npn \@@_special_W:
2789 {
2790     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2791     { \@@_warning:n { W~warning } }
2792 }

```

For S (of siunitx).

```

2793 \AtBeginDocument
2794 {
2795     \IfPackageLoadedT { siunitx }
2796     {
2797         \cs_new_protected:Npn \@@_S: #1 #2

```

```

2798         {
2799             \str_if_eq:nnTF { #2 } { [ ] }
2800             { \@@_make_preamble_S:w [ ] }
2801             { \@@_make_preamble_S:w [ ] { #2 } }
2802         }
2803     }
2804 }

2805 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2806 { \@@_make_preamble_S_i:n { #1 } }

2807 \cs_new_protected:Npn \@@_test_siunitx:
2808 {
2809     \IfPackageAtLeastF { siunitx } { 2026/03/26 }
2810     { \@@_fatal:n { siunitx~too~old } }
2811     \cs_gset_eq:NN \@@_test_siunitx: \prg_do_nothing:
2812 }
2813 % \end{macrocode}
2814 %
2815 % \begin{macrocode}
2816 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2817 {
2818     \@@_test_siunitx:
2819     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2820     \tl_gclear:N \g_@@_pre_cell_tl
2821     \tl_gput_right:Nn \g_@@_array_preamble_tl
2822     {
2823         > {

```

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).

```

2824         \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2825         \keys_set:nn { siunitx } { #1 }
2826         \@@_cell_begin:
2827         \siunitx_cell_begin:w
2828     }
2829     c
2830     <
2831     {
2832         \siunitx_cell_end:

```

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

```

2833         \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2834         {
2835             \bool_if:NTF \l__siunitx_table_text_bool
2836             \bool_set_true:N
2837             \bool_set_false:N
2838             \l__siunitx_table_text_bool
2839         }
2840         \@@_cell_end:
2841     }
2842 }

```

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

```

2843     \int_gincr:N \c@jCol
2844     \@@_rec_preamble_after_col:n
2845 }

```

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

```

2846 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2

```

```

2847 {
2848   \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.

```

2849   \int_if_zero:nTF \c@jCol
2850   {
2851     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2852     {

```

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

```

2853       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2854       \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2855       \@@_rec_preamble:n #2
2856     }
2857     {
2858       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2859       \@@_make_preamble_iv:nn { #1 } { #2 }
2860     }
2861   }
2862   { \@@_make_preamble_iv:nn { #1 } { #2 } }
2863 }
2864 \cs_set_eq:cc { @@ _ \token_to_str:N [ : ] { @@ _ \token_to_str:N ( : }
2865 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2866 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2867 {
2868   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2869   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2870   \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2871   {
2872     \@@_error:nn { delimiter~after~opening } { #2 }
2873     \@@_rec_preamble:n
2874   }
2875   { \@@_rec_preamble:n #2 }
2876 }

```

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

```

2877 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2878 { \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}).

```

2879 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2880 {
2881   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2882   \tl_if_in:nnTF { ) ] \} } { #2 }
2883   { \@@_make_preamble_v:nnn #1 #2 }
2884   {
2885     \str_if_eq:nnTF \s_stop { #2 }
2886     {
2887       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2888       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2889       {
2890         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2891         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2892         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2893         \@@_rec_preamble:n #2
2894       }
2895     }
2896     {
2897       \tl_if_in:nnT { ( [ \{ \left } { #2 }

```

```

2898         { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2899         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2900         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2901         \@@_rec_preamble:n #2
2902     }
2903 }
2904 }
2905 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2906 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2907 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2908 {
2909     \str_if_eq:nnTF \s_stop { #3 }
2910     {
2911         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2912         {
2913             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2914             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2915             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2916             \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2917         }
2918         {
2919             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2920             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2921             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2922             \@@_error:nn { double~closing~delimiter } { #2 }
2923         }
2924     }
2925     {
2926         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2927         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2928         \@@_error:nn { double~closing~delimiter } { #2 }
2929         \@@_rec_preamble:n #3
2930     }
2931 }
2932 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2933 { \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 @{...}.

```

2934 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2935 {
2936     \str_if_eq:nnTF { #1 } { < }
2937     { \@@_rec_preamble_after_col_i:n }
2938     {
2939         \str_if_eq:nnTF { #1 } { @ }
2940         { \@@_rec_preamble_after_col_ii:n }
2941         {
2942             \str_if_eq:eeTF \l_@@_vlines_clist { all }
2943             {
2944                 \tl_gput_right:Nn \g_@@_array_preamble_tl
2945                 { ! { \skip_horizontal:N \arrayrulewidth } }
2946             }
2947             {
2948                 \clist_if_in:NeT \l_@@_vlines_clist
2949                 { \int_eval:n { \c@jCol + 1 } }
2950                 {
2951                     \tl_gput_right:Nn \g_@@_array_preamble_tl
2952                     { ! { \skip_horizontal:N \arrayrulewidth } }
2953                 }
2954             }
2955         }
2956     }
2957 }

```

```

2955         \@@_rec_preamble:n { #1 }
2956     }
2957 }
2958 }
2959 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2960 {
2961     \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2962     \@@_rec_preamble_after_col:n
2963 }

```

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.

```

2964 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2965 {
2966     \str_if_eq:eeTF \l_@@_vlines_clist { all }
2967     {
2968         \tl_gput_right:Nn \g_@@_array_preamble_tl
2969         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2970     }
2971     {
2972         \clist_if_in:NcTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2973         {
2974             \tl_gput_right:Nn \g_@@_array_preamble_tl
2975             { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2976         }
2977         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2978     }
2979     \@@_rec_preamble:n
2980 }
2981 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2982 {
2983     \tl_clear:N \l_tmpa_tl
2984     \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2985     \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2986 }

```

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.

```

2987 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2988 { \@@_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.

```

2989 \cs_new_protected:Npn \@@_X: #1 #2
2990 {
2991     \str_if_eq:nnTF { #2 } { [ ]
2992     { \@@_make_preamble_X:w [ ]
2993     { \@@_make_preamble_X:w [ ] #2 }
2994     }
2995     \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2996     { \@@_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.

```

2997 \keys_define:nn { nicematrix / X-column }
2998 {

```

```

2999 V .code:n =
3000   \IfPackageLoadedTF { varwidth }
3001   {
3002     \bool_set_true:N \l_@@_V_of_X_bool
3003     \bool_gset_true:N \g_@@_V_of_X_bool
3004   }
3005   { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
3006   unknown .code:n =
3007     \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
3008     { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
3009     { \@@_error_or_warning:n { invalid~weight } }
3010 }

```

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

```

3011 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
3012 {

```

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).

```

3013   \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).

```

3014   \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`).

```

3015   \fp_set:Nn \l_tmpa_fp { 1.0 }
3016   \@@_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.

```

3017   \bool_set_false:N \l_@@_V_of_X_bool
3018   \keys_set:no { nicematrix / X-column } \l_tmpa_tl

```

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

```

3019   \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).

```

3020   \bool_if:NTF \l_@@_X_columns_aux_bool
3021   {
3022     \@@_make_preamble_ii_iv:nnn

```

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

```

3023     { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
3024     { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
3025     { \@@_no_update_width: }
3026   }

```

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.

```

3027   {
3028     \tl_gput_right:Nn \g_@@_array_preamble_tl
3029     {
3030       > {
3031         \@@_cell_begin:
3032         \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`.

```

3033   \NotEmpty

```

The following code will nullify the box of the cell.

```
3034         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
3035         { \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`.

```
3036         \begin { minipage } { 5 cm } \arraybackslash
3037     }
3038     c
3039     < {
3040         \end { minipage }
3041         \@@_cell_end:
3042     }
3043 }
3044 \int_gincr:N \c@jCol
3045 \@@_rec_preamble_after_col:n
3046 }
3047 }

3048 \cs_new_protected:Npn \@@_no_update_width:
3049 {
3050     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
3051     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
3052 }
```

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

```
3053 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
3054 {
3055     \seq_gput_right:Ne \g_@@_cols_vlism_seq
3056     { \int_eval:n { \c@jCol + 1 } }
3057     \tl_gput_right:Ne \g_@@_array_preamble_tl
3058     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
3059     \@@_rec_preamble:n
3060 }
```

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.

```
3061 \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).

```
3062 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
3063 { \@@_fatal:n { Preamble-forgotten } }
3064 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
3065 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
3066 { @@ _ \token_to_str:N \hline : }
3067 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
3068 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
3069 { @@ _ \token_to_str:N \hline : }
3070 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
3071 { @@ _ \token_to_str:N \hline : }
3072 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
3073 { @@ _ \token_to_str:N \hline : }
3074 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
3075 { @@ _ \token_to_str:N \hline : }
3076 \cs_new_protected:cpn { @@ _ \token_to_str:N \linewidth : }
3077 { \@@_fatal:n { NiceTabularX-probably-required } }
3078 \cs_set_eq:cc { @@ _ \token_to_str:N \textwidth : }
3079 { @@ _ \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).

```
3080 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
3081 {
```

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`.

```
3082 \multispan { #1 }
3083 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
3084 \begingroup
3085 \tbl_update_multicolumn_cell_data:n { #1 }
```

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

```
3086 \tl_gclear:N \g_@@_preamble_tl
3087 \@@_make_m_preamble:n #2 \q_stop
```

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

```
3088 \def \@addamp
3089 {
3090 \legacy_if:nTF { @firstamp }
3091 { \legacy_if_set_false:n { @firstamp } }
3092 { \@preamerr 5 }
3093 }
3094 \exp_args:No \@mkpream \g_@@_preamble_tl
3095 \@addtopreamble \@empty
3096 \endgroup
3097 \UseTaggingSocket { tbl / colspan } { #1 }
```

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

```
3098 \int_compare:nNnT { #1 } > 1
3099 {
3100 \seq_gput_right:Ne \g_@@_multicolumn_cells_seq
3101 { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
3102 \seq_gput_right:Nn \g_@@_multicolumn_sizes_seq { #1 }
3103 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
3104 {
3105 {
3106 \int_if_zero:nTF \c@jCol
3107 { \int_eval:n { \c@iRow + 1 } }
3108 { \int_use:N \c@iRow }
3109 }
3110 { \int_eval:n { \c@jCol + 1 } }
3111 {
3112 \int_if_zero:nTF \c@jCol
3113 { \int_eval:n { \c@iRow + 1 } }
3114 { \int_use:N \c@iRow }
3115 }
3116 { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block.

```
3117 { }
3118 }
3119 }
```

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

```
3120 \RenewDocumentCommand { \cellcolor } { 0 { } m }
3121 {
```

```

3122     \tl_gput_right:Nc \g_@@_pre_code_before_tl
3123     {
3124         \@@_rectanglecolor [ ##1 ]
3125         { \exp_not:n { ##2 } }
3126         { \int_use:N \c@iRow - \int_use:N \c@jCol }
3127         { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
3128     }
3129     \ignorespaces
3130 }

```

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

```

3131     \def \@sharp { #3 }
3132     \@arstrut
3133     \@preamble
3134     \null

```

We add some lines.

```

3135     \int_gadd:Nn \c@jCol { #1 - 1 }
3136     \int_compare:nNnT \c@jCol > \g_@@_col_total_int
3137     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3138     \ignorespaces
3139 }

```

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`.

```

3140 \
3141 \cs_new_protected:Npn \@@_make_m_preamble:n #1
3142 {
3143     \str_case:nnF { #1 }
3144     {
3145         c { \@@_make_m_preamble_i:n #1 }
3146         l { \@@_make_m_preamble_i:n #1 }
3147         X { \@@_make_m_preamble_i:n l }
3148         r { \@@_make_m_preamble_i:n #1 }
3149         > { \@@_make_m_preamble_ii:nn #1 }
3150         ! { \@@_make_m_preamble_ii:nn #1 }
3151         @ { \@@_make_m_preamble_ii:nn #1 }
3152         | { \@@_make_m_preamble_iii:n #1 }
3153         p { \@@_make_m_preamble_iv:nnn t #1 }
3154         m { \@@_make_m_preamble_iv:nnn c #1 }
3155         b { \@@_make_m_preamble_iv:nnn b #1 }
3156         w { \@@_make_m_preamble_v:nnnn { } #1 }
3157         W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
3158         \q_stop { }
3159     }
3160     {
3161         \cs_if_exist:cTF { NC @ find @ #1 }
3162         {
3163             \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
3164             \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
3165         }
3166         {
3167             \str_if_eq:nnTF { #1 } { S }
3168             { \@@_fatal:n { unknown~column~type~S~multicolumn } }
3169             { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
3170         }
3171     }
3172 }

```

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

```

3173 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
3174 {

```

```

3175 \tl_gput_right:Nn \g_@@_preamble_tl
3176 {
3177   > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
3178   #1
3179   < \@@_cell_end:
3180 }

```

We test for the presence of a <.

```

3181 \@@_make_m_preamble_x:n
3182 }

```

For >, ! and @

```

3183 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
3184 {
3185   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
3186   \@@_make_m_preamble:n
3187 }

```

For |

```

3188 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
3189 {
3190   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
3191   \@@_make_m_preamble:n
3192 }

```

For p, m and b

```

3193 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
3194 {
3195   \tl_gput_right:Nn \g_@@_preamble_tl
3196   {
3197     > {
3198       \@@_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`.

```

3199 \setlength { \l_tmpa_dim } { #3 }
3200 \begin { minipage } [ #1 ] { \l_tmpa_dim }
3201 \mode_leave_vertical:
3202 \arraybackslash
3203 \vrule height \box_ht:N \@arstrutbox depth \c_zero_dim width \c_zero_dim
3204 }
3205 c
3206 < {
3207   \vrule height \c_zero_dim depth \box_dp:N \@arstrutbox width \c_zero_dim
3208   \end { minipage }
3209   \@@_cell_end:
3210 }
3211 }

```

We test for the presence of a <.

```

3212 \@@_make_m_preamble_x:n
3213 }

```

For w and W

```

3214 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3215 {
3216   \tl_gput_right:Nn \g_@@_preamble_tl
3217   {
3218     > {
3219       \dim_set:Nn \l_@@_col_width_dim { #4 }
3220       \hbox_set:Nw \l_@@_cell_box
3221       \@@_cell_begin:
3222       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }

```

```

3223     }
3224     c
3225     < {
3226         \@@_cell_end:
3227         \hbox_set_end:
3228         \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3229         #1
3230         \@@_adjust_size_box:
3231         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3232     }
3233 }

```

We test for the presence of a <.

```

3234     \@@_make_m_preamble_x:n
3235 }

```

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

```

3236 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3237 {
3238     \str_if_eq:nnTF { #1 } { < }
3239     \@@_make_m_preamble_ix:n
3240     { \@@_make_m_preamble:n { #1 } }
3241 }
3242 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3243 {
3244     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3245     \@@_make_m_preamble_x:n
3246 }

```

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`).

```

3247 \cs_new_protected:Npn \@@_put_box_in_flow:
3248 {
3249     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3250     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3251     \str_if_eq:eeTF \l_@@_baseline_tl { c }
3252     { \box_use_drop:N \l_tmpa_box }
3253     \@@_put_box_in_flow_i:
3254 }

```

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

```

3255 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3256 {
3257     \pgfpicture
3258     \@@_qpoint:n { row - 1 }
3259     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3260     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3261     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3262     \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).

```

3263     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3264     {
3265         \int_set:Nn \l_tmpa_int
3266         { \str_range:Nnn \l_@@_baseline_tl { 6 } { -1 } }
3267         \bool_lazy_or:nnT
3268         { \int_compare_p:nNn \l_tmpa_int < { 1 } }
3269         { \int_compare_p:nNn \l_tmpa_int > { \c@iRow + 1 } }

```

```

3270     {
3271       \@@_error:n { bad-value-for-baseline-line }
3272       \int_set:Nn \l_tmpa_int 1
3273     }
3274     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3275   }
3276   {
3277     \str_if_eq:eeTF t \l_@@_baseline_tl
3278     { \int_set:Nn \l_tmpa_int 1 }
3279     {
3280       \str_if_eq:eeTF b \l_@@_baseline_tl
3281       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3282       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3283     }
3284     \bool_lazy_or:nnT
3285     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3286     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3287     {
3288       \@@_error:n { bad-value-for-baseline }
3289       \int_set:Nn \l_tmpa_int 1
3290     }
3291     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3292     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3293   }
3294   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3295   \endpgfpicture
3296   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3297 }

```

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).

```

3298 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3299 {

```

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.

```

3300   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3301   {
3302     \int_compare:nNnT \c@jCol > 1
3303     {
3304       \box_set_wd:Nn \l_@@_the_array_box
3305       { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3306     }
3307   }

```

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

```

3308   \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3309   \bool_if:NT \l_@@_caption_above_bool
3310   {
3311     \tl_if_empty:NF \l_@@_caption_tl
3312     {
3313       \bool_set_false:N \g_@@_caption_finished_bool
3314       \int_gzero:N \c@tabularnote
3315       \@@_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).

```

3316         \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3317         {
3318             \tl_gput_right:Ne \g_@@_aux_tl
3319             {
3320                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3321                 { \int_use:N \g_@@_notes_caption_int }
3322             }
3323             \int_gzero:N \g_@@_notes_caption_int
3324         }
3325     }
3326 }

```

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

```

3327     \hbox
3328     {
3329         \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.

```

3330         \@@_create_extra_nodes:
3331         \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3332     }

```

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).

```

3333     \bool_lazy_any:nT
3334     {
3335         { ! \seq_if_empty_p:N \g_@@_notes_seq }
3336         { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3337         { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3338     }
3339     {
3340         \bool_if:NTF \l_@@_notes_no_print_bool
3341         { \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes: }
3342         \@@_tabular_notes:
3343     }
3344     \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3345     \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3346     \end { minipage }
3347 }

```

```

3348 \cs_new_protected:Npn \@@_insert_caption:
3349 {
3350     \tl_if_empty:NF \l_@@_caption_tl
3351     {
3352         \cs_if_exist:NTF \@capttype
3353         { \@@_insert_caption_i: }
3354         { \@@_error:n { caption~outside~float } }
3355     }
3356 }

```

```

3357 \cs_new_protected:Npn \@@_insert_caption_i:
3358 {
3359     \group_begin:

```

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

```
3360 \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.

```
3361 \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
3362 \tl_if_empty:NTF \l_@@_short_caption_tl
3363 \caption
3364 { \caption [ \l_@@_short_caption_tl ] }
3365 { \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.

```
3366 \bool_if:NF \g_@@_caption_finished_bool
3367 {
3368   \bool_gset_true:N \g_@@_caption_finished_bool
3369   \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3370   \int_gzero:N \c@tabularnote
3371 }
3372 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3373 \group_end:
3374 }

3375 \cs_new_protected:Npn \@_tabularnote_error:n #1
3376 {
3377   \@_error_or_warning:n { tabularnote~below~the~tabular }
3378   \cs_gset:Npn \@_tabularnote_error:n ##1 { }
3379 }

3380 \cs_set_protected:Npn \@_tabular_notes_error:
3381 { \@_error:n { Misuse~of~NiceTabularNotes } }

3382 \cs_set_eq:NN \NiceTabularNotes \@_tabular_notes_error:

3383 \cs_set_protected:Npn \@_tabular_notes:
3384 {
3385   \cs_gset_eq:NN \NiceTabularNotes \@_tabular_notes_error:
3386   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3387   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3388   \skip_vertical:N 0.65ex
```

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

```
3389 \group_begin:
3390 \l_@@_notes_code_before_tl
3391 \tl_if_empty:NF \g_@@_tabularnote_tl
3392 {
3393   \g_@@_tabularnote_tl \par
3394   \tl_gclear:N \g_@@_tabularnote_tl
3395 }
```

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.

```
3396 \int_compare:nNnT \c@tabularnote > \c_zero_int
3397 {
3398   \bool_if:NTF \l_@@_notes_para_bool
3399   {
3400     \begin { tabularnotes* }
3401     \seq_map_inline:Nn \g_@@_notes_seq
3402       { \@_one_tabularnote:nn ##1 }
3403     \strut
3404     \end { tabularnotes* }
```

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

```

3405         \par
3406     }
3407     {
3408         \tabularnotes
3409         \seq_map_inline:Nn \g_@@_notes_seq
3410         { \@@_one_tabularnote:nn ##1 }
3411         \strut
3412         \endtabularnotes
3413     }
3414 }
3415 \unskip
3416 \group_end:
3417 \bool_if:NT \l_@@_notes_bottomrule_bool
3418 {
3419     \IfPackageLoadedTF { booktabs }
3420     {

```

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

```

3421         \skip_vertical:N \aboverulesep

```

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

```

3422         { \CT@arc@ \hrule height \heavyrulewidth }
3423     }
3424     { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3425 }
3426 \l_@@_notes_code_after_tl
3427 \seq_gclear:N \g_@@_notes_seq
3428 \seq_gclear:N \g_@@_notes_in_caption_seq
3429 \int_gzero:N \c@tabularnote
3430 }

```

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

```

3431 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3432 {
3433     \tl_if_novalue:nTF { #1 }
3434     { \item }
3435     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3436 }

```

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`.

```

3437 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3438 {
3439     \pgfpicture
3440     \@@_qpoint:n { row - 1 }
3441     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3442     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3443     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3444     \endpgfpicture
3445     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3446     \int_if_zero:nT \l_@@_first_row_int
3447     {
3448         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3449         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3450     }
3451     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3452 }

```


Now, the general case.

```

3453 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3454 {

```

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

```

3455 \str_if_eq:eeT \l_@@_baseline_tl { t }
3456 { \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`.

```

3457 \pgfpicture
3458 \@@_qpoint:n { row - 1 }
3459 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3460 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3461 {
3462   \int_set:Nn \l_tmpa_int
3463   {
3464     \str_range:Nnn
3465     \l_@@_baseline_tl
3466     { 6 }
3467     { \tl_count:o \l_@@_baseline_tl }
3468   }
3469   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3470 }
3471 {
3472   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3473   \bool_lazy_or:nnT
3474   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3475   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3476   {
3477     \@@_error:n { bad~value~for~baseline }
3478     \int_set:Nn \l_tmpa_int 1
3479   }
3480   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3481 }
3482 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3483 \endpgfpicture
3484 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3485 \int_if_zero:nT \l_@@_first_row_int
3486 {
3487   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3488   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3489 }
3490 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3491 }

```

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.

```

3492 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3493 {

```

We will compute the real width of both delimiters used.

```

3494 \dim_zero_new:N \l_@@_real_left_delim_dim
3495 \dim_zero_new:N \l_@@_real_right_delim_dim
3496 \hbox_set:Nn \l_tmpb_box
3497 {
3498   \m@th
3499   $ % $
3500   \left #1
3501   \vcenter
3502   {
3503     \vbox_to_ht:nn
3504     { \box_ht_plus_dp:N \l_tmpa_box }

```

```

3505         { }
3506     }
3507     \right .
3508     $ % $
3509 }
3510 \dim_set:Nn \l_@@_real_left_delim_dim
3511 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3512 \hbox_set:Nn \l_tmpb_box
3513 {
3514     \m@th
3515     $ % $
3516     \left .
3517     \vbox_to_ht:nn
3518     { \box_ht_plus_dp:N \l_tmpa_box }
3519     { }
3520     \right #2
3521     $ % $
3522 }
3523 \dim_set:Nn \l_@@_real_right_delim_dim
3524 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3525     \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3526     \@@_put_box_in_flow:
3527     \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3528 }

```

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).

```

3529 \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).

```

3530 {
3531     \peek_remove_spaces:n
3532     {
3533         \peek_meaning:NTF \end
3534         \@@_analyze_end:Nn
3535         {
3536             \@@_transform_preamble:
3537             \@@_array:o \g_@@_array_preamble_tl
3538         }
3539     }
3540 }
3541 {
3542     \@@_create_col_nodes:
3543     \endarray
3544 }

```

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

```

3545 \NewDocumentEnvironment { @@-light-syntax } { b }
3546 {

```

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`.

```

3547     \tl_if_empty:nT { #1 }

```

```

3548     { \@@_fatal:n { empty-environment } }
3549 \tl_if_in:nnT { #1 } { & }
3550     { \@@_fatal:n { ampersand-in~light-syntax } }
3551 \tl_if_in:nnT { #1 } { \ }
3552     { \@@_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`.

```

3553 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3554 }

```

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.

```

3555 {
3556   \@@_create_col_nodes:
3557   \endarray
3558 }
3559 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3560 {
3561   \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).

```

3562 \seq_clear_new:N \l_@@_rows_seq

```

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

```

3563 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3564 \bool_if:NTF \l_@@_light_syntax_expanded_bool
3565   \seq_set_split:Nee
3566   \seq_set_split:Non
3567   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3568 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3569 \tl_if_empty:NF \l_tmpa_tl
3570 { \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.

```

3571 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3572 { \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`.

```

3573 \tl_build_begin:N \l_@@_new_body_tl
3574 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3575 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3576 \@@_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).

```

3577 \seq_map_inline:Nn \l_@@_rows_seq
3578 {
3579   \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3580   \@@_line_with_light_syntax:n { ##1 }
3581 }
3582 \tl_build_end:N \l_@@_new_body_tl

```

```

3583 \int_compare:nNtT \l_@@_last_col_int = { -1 }
3584 {
3585     \int_set:Nn \l_@@_last_col_int
3586     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3587 }

```

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.

```

3588 \@@_transform_preamble:

3589 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3590 }

3591 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3592 {
3593     \seq_clear_new:N \l_@@_cells_seq
3594     \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3595     \int_set:Nn \l_@@_nb_cols_int
3596     { \int_max:nn \l_@@_nb_cols_int { \seq_count:N \l_@@_cells_seq } }
3597     \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3598     \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3599     \seq_map_inline:Nn \l_@@_cells_seq
3600     { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3601 }
3602 \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`.

```

3603 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3604 {
3605     \str_if_eq:eeT \g_@@_name_env_str { #2 }
3606     { \@@_fatal:n { empty~environment } }

```

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

```

3607 \end { #2 }
3608 }

```

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`).

```

3609 \cs_new:Npn \@@_create_col_nodes:
3610 {
3611     \crrc
3612     \int_if_zero:nT \l_@@_first_col_int
3613     {
3614         \omit
3615         \hbox_overlap_left:n
3616         {
3617             \bool_if:NT \l_@@_code_before_bool
3618             { \pgfsys@markposition { \@@_env: - col - 0 } }
3619             \pgfpicture
3620             \pgfrememberpicturepositiononpagetrue
3621             \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3622             \str_if_empty:NF \l_@@_name_str
3623             { \pgfnodelalias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3624             \endpgfpicture
3625             \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3626         }
3627         &
3628     }
3629     \omit

```

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

```
3630 \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`).

```
3631 \int_if_zero:nTF \l_@@_first_col_int
3632 {
3633   \@@_mark_position:n { 1 }
3634   \pgfpicture
3635   \pgfrememberpicturepositiononpagetrue
3636   \pgfcoordinate { \@@_env: - col - 1 }
3637   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3638   \str_if_empty:NF \l_@@_name_str
3639   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3640   \endpgfpicture
3641 }
3642 {
3643   \bool_if:NT \l_@@_code_before_bool
3644   {
3645     \hbox
3646     {
3647       \skip_horizontal:n { 0.5 \arrayrulewidth }
3648       \pgfsys@markposition { \@@_env: - col - 1 }
3649       \skip_horizontal:n { -0.5 \arrayrulewidth }
3650     }
3651   }
3652   \pgfpicture
3653   \pgfrememberpicturepositiononpagetrue
3654   \pgfcoordinate { \@@_env: - col - 1 }
3655   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3656   \@@_node_alias:n { 1 }
3657   \endpgfpicture
3658 }
```

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.

```
3659 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3660 \bool_if:NF \l_@@_auto_columns_width_bool
3661 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3662 {
3663   \bool_lazy_and:nnTF
3664   \l_@@_auto_columns_width_bool
3665   { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3666   { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3667   { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3668   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3669 }
3670 \skip_horizontal:N \g_tmpa_skip
3671 \hbox
3672 {
3673   \@@_mark_position:n { 2 }
3674   \pgfpicture
3675   \pgfrememberpicturepositiononpagetrue
3676   \pgfcoordinate { \@@_env: - col - 2 }
3677   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3678   \@@_node_alias:n { 2 }
3679   \endpgfpicture
3680 }
```

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.

```

3681 \int_gset:Nn \g_tmpa_int 1
3682 \bool_if:NTF \g_@@_last_col_found_bool
3683 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3684 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3685 {
3686   &
3687   \omit
3688   \int_gincr:N \g_tmpa_int

```

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

```

3689 \skip_horizontal:N \g_tmpa_skip
3690 \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

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

```

3691 \pgfpicture
3692 \pgfrememberpicturepositiononpagetrue
3693 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3694 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3695 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3696 \endpgfpicture
3697 }

```

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).

```

3698 \bool_lazy_or:nnF
3699 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3700 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3701 {
3702   &
3703   \omit
3704   \skip_horizontal:N \g_tmpa_skip
3705   \int_gincr:N \g_tmpa_int
3706   \bool_lazy_any:nF
3707   {
3708     \g_@@_delims_bool
3709     \l_@@_tabular_bool
3710     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3711     \l_@@_exterior_arraycolsep_bool
3712     \l_@@_bar_at_end_of_pream_bool
3713   }
3714   { \skip_horizontal:n { - \col@sep } }
3715   \bool_if:NT \l_@@_code_before_bool
3716   {
3717     \hbox
3718     {
3719       \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.

```

3720 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3721 { \skip_horizontal:n { - \arraycolsep } }
3722 \pgfsys@markposition
3723 { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3724 \skip_horizontal:n { 0.5 \arrayrulewidth }
3725 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3726 { \skip_horizontal:N \arraycolsep }
3727 }
3728 }
3729 \pgfpicture
3730 \pgfrememberpicturepositiononpagetrue
3731 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3732 {

```

```

3733         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3734         {
3735             \pgfpoint
3736             { - 0.5 \arrayrulewidth - \arraycolsep }
3737             \c_zero_dim
3738         }
3739         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3740     }
3741     \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3742 \endpgfpicture
3743 }

3744 \bool_if:NT \g_@@_last_col_found_bool
3745 {
3746     \hbox_overlap_right:n
3747     {
3748         \skip_horizontal:N \g_@@_width_last_col_dim
3749         \skip_horizontal:N \colsep
3750         \bool_if:NT \l_@@_code_before_bool
3751         {
3752             \pgfsys@markposition
3753             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3754         }
3755         \pgfpicture
3756         \pgfrememberpicturepositiononpagetrue
3757         \pgfcoordinate
3758         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3759         \pgfpintorigin
3760         \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3761         \endpgfpicture
3762     }
3763 }
3764 }

3765 \cs_new_protected:Npn \@@_mark_position:n #1
3766 {
3767     \bool_if:NT \l_@@_code_before_bool
3768     {
3769         \hbox
3770         {
3771             \skip_horizontal:n { -0.5 \arrayrulewidth }
3772             \pgfsys@markposition { \@@_env: - col - #1 }
3773             \skip_horizontal:n { 0.5 \arrayrulewidth }
3774         }
3775     }
3776 }

3777 \cs_new_protected:Npn \@@_node_alias:n #1
3778 {
3779     \str_if_empty:NF \l_@@_name_str
3780     { \pgfnodelias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3781 }

```

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

```

3782 \tl_const:Nn \c_@@_preamble_first_col_tl
3783 {
3784     >
3785     {

```

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

```

3786      \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3787      \cs_set_eq:NN \Hline \@@_Hline_in_cell:
3788      \cs_set_eq:NN \FalseRow \@@_FalseRow_in_cell: % added 2026-07-17
3789      \bool_gset_true:N \g_@@_after_col_zero_bool
3790      \@@_begin_of_row:
3791      \hbox_set:Nw \l_@@_cell_box
3792      \@@_math_toggle:
3793      \@@_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”.

```

3794      \int_compare:nNnT \c@iRow > \c_zero_int
3795      {
3796        \bool_lazy_or:nnT
3797        { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3798        { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3799        {
3800          \l_@@_code_for_first_col_tl
3801          \xglobal \colorlet { nicematrix-first-col } { . }
3802        }
3803      }
3804    }

```

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`.

```

3805      l
3806      <
3807      {
3808        \@@_math_toggle:
3809        \hbox_set_end:
3810        \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3811        \@@_adjust_size_box:
3812        \@@_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.

```

3813      \dim_gset:Nn \g_@@_width_first_col_dim
3814      { \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.

```

3815      \hbox_overlap_left:n
3816      {
3817        \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3818        \@@_node_cell:
3819        { \box_use_drop:N \l_@@_cell_box }
3820        \skip_horizontal:N \l_@@_left_delim_dim
3821        \skip_horizontal:N \l_@@_left_margin_dim
3822        \skip_horizontal:N \l_@@_extra_left_margin_dim
3823      }
3824      \bool_gset_false:N \g_@@_empty_cell_bool
3825      \skip_horizontal:n { -2 \col@sep }
3826    }
3827  }

```

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

```

3828      \tl_const:Nn \c_@@_preamble_last_col_tl
3829      {
3830        >
3831        {
3832          \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).

```
3833 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3834 \cs_set_eq:NN \Hline \@@_Hline_in_cell:
3835 \cs_set_eq:NN \FalseRow \@@_FalseRow_in_cell: % added 2026-07-17
```

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

```
3836 \bool_gset_true:N \g_@@_last_col_found_bool
3837 \int_gincr:N \c@jCol
3838 \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3839 \hbox_set:Nw \l_@@_cell_box
3840 \@@_math_toggle:
3841 \@@_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”.

```
3842 \int_compare:nNnT \c@iRow > \c_zero_int
3843 {
3844   \bool_lazy_or:nnT
3845   { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3846   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3847   {
3848     \l_@@_code_for_last_col_tl
3849     \xglobal \colorlet { nicematrix-last-col } { . }
3850   }
3851 }
3852 }
3853 1
3854 <
3855 {
3856   \@@_math_toggle:
3857   \hbox_set_end:
3858   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3859   \@@_adjust_size_box:
3860   \@@_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.

```
3861 \dim_gset:Nn \g_@@_width_last_col_dim
3862 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3863 \skip_horizontal:n { -2 \col@sep }
```

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

```
3864 \hbox_overlap_right:n
3865 {
3866   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3867   {
3868     \skip_horizontal:N \l_@@_right_delim_dim
3869     \skip_horizontal:N \l_@@_right_margin_dim
3870     \skip_horizontal:N \l_@@_extra_right_margin_dim
3871     \@@_node_cell:
3872   }
3873 }
3874 \bool_gset_false:N \g_@@_empty_cell_bool
3875 }
3876 }
```

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

```
3877 \NewDocumentEnvironment { NiceArray } { }
3878 {
3879   \bool_gset_false:N \g_@@_delims_bool
3880   \str_if_empty:NT \g_@@_name_env_str
3881   { \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).

```

3882   \NiceArrayWithDelims . .
3883 }
3884 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

3885 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3886 {
3887   \NewDocumentEnvironment { #1 NiceArray } { }
3888   {
3889     \bool_gset_true:N \g_@@_delims_bool
3890     \str_if_empty:NT \g_@@_name_env_str
3891       { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3892     \@@_test_if_math_mode:
3893     \NiceArrayWithDelims #2 #3
3894   }
3895   { \endNiceArrayWithDelims }
3896 }
3897 \@@_def_env:NNN p ( )
3898 \@@_def_env:NNN b [ ]
3899 \@@_def_env:NNN B \{ \}
3900 \@@_def_env:NNN v \vert \vert
3901 \@@_def_env:NNN V \Vert \Vert

```

13 The environment {NiceMatrix} and its variants

13.1 Definition of {pNiceMatrix}

```

3902 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3903 {
3904   \bool_set_false:N \l_@@_preamble_bool
3905   \tl_clear:N \l_tmpa_tl
3906   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3907     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3908   \tl_put_right:Nn \l_tmpa_tl
3909     {
3910       *
3911       {
3912         \int_case:nnF \l_@@_last_col_int
3913         {
3914           { -2 } { \c@MaxMatrixCols }
3915           { -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).

```

3916       }
3917       { \int_eval:n { \l_@@_last_col_int - 1 } }
3918     }
3919     { #2 }
3920   }
3921   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3922   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3923 }
3924 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3925 \clist_map_inline:nn { p , b , B , v , V }
3926 {
3927   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }

```

```

3928 {
3929   \bool_gset_true:N \g_@@_delims_bool
3930   \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3931   \int_if_zero:nT \l_@@_last_col_int
3932   {
3933     \bool_set_true:N \l_@@_last_col_without_value_bool
3934     \int_set:Nn \l_@@_last_col_int { -1 }
3935   }
3936   \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3937   \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3938 }
3939 { \use:c { end #1 NiceArray } }
3940 }

```

We define also an environment {NiceMatrix}

```

3941 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3942 {
3943   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3944   \int_if_zero:nT \l_@@_last_col_int
3945   {
3946     \bool_set_true:N \l_@@_last_col_without_value_bool
3947     \int_set:Nn \l_@@_last_col_int { -1 }
3948   }
3949   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3950   \bool_lazy_or:nnT
3951     \l_@@_except_borders_bool
3952     { \clist_if_empty_p:N \l_@@_vlines_clist }
3953     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3954   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3955 }
3956 { \endNiceArray }

```

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

```

3957 \cs_new_protected:Npn \@@_NotEmpty:
3958 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

13.2 The key renew-matrix

```

3959 \cs_set_protected:Npn \@@_renew_matrix:
3960 {
3961   \tl_map_inline:nn { pvVbB }
3962   { \RenewEnvironmentCopy { ##1matrix } { ##1NiceMatrix } }
3963 }

```

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

```

3964 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3965 {

```

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.

```

3966   \dim_compare:nNnT\l_@@_width_dim = \c_zero_dim
3967   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3968   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3969   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3970   \tl_if_empty:NF \l_@@_short_caption_tl
3971   {
3972     \tl_if_empty:NT \l_@@_caption_tl
3973     {
3974       \@@_error_or_warning:n { short-caption-without~caption }
3975       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl

```

```

3976     }
3977   }
3978   \tl_if_empty:NF \l_@@_label_tl
3979   {
3980     \tl_if_empty:NT \l_@@_caption_tl
3981     { \@@_error_or_warning:n { label~without~caption } }
3982   }
3983   \NewDocumentEnvironment { TabularNote } { b }
3984   {
3985     \bool_if:NTF \l_@@_in_code_after_bool
3986     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3987     {
3988       \tl_if_empty:NF \g_@@_tabularnote_tl
3989       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3990       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3991     }
3992   }
3993   { }
3994   \@@_settings_for_tabular:
3995   \NiceArray { #2 }
3996 }
3997 { \endNiceArray }
3998 \cs_new_protected:Npn \@@_settings_for_tabular:
3999 {
4000   \bool_set_true:N \l_@@_tabular_bool
4001   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
4002   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
4003   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
4004 }

4005 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
4006 {
4007   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
4008   \dim_set:Nn \l_@@_width_dim { #1 }
4009   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
4010   \@@_settings_for_tabular:
4011   \NiceArray { #3 }
4012 }
4013 {
4014   \endNiceArray
4015   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
4016   { \@@_error:n { NiceTabularX~without~X } }
4017 }

4018 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
4019 {
4020   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
4021   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
4022   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
4023   \@@_settings_for_tabular:
4024   \NiceArray { #3 }
4025 }
4026 { \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`).

```

4027 \cs_new_protected:Npn \l_@@_deal_with_rounded_corners:
4028 {
4029   \bool_lazy_all:nT
4030   {
4031     { \dim_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
4032     { \l_@@_hvlines_bool }
4033     { ! \g_@@_delims_bool }
4034     { ! \l_@@_except_borders_bool }
4035   }
4036   {
4037     \bool_set_true:N \l_@@_except_borders_bool
4038     \clist_if_empty:NF \l_@@_corners_clist
4039     { \@@_error:n { hvlines,~rounded-corners-and-corners } }
4040     \tl_gput_right:Nn \g_@@_rules_tl
4041     {
4042       \@@_stroke_block:nnnnn
4043       {
4044         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
4045         draw = \l_@@_rules_color_tl
4046       }
4047       { 1 } { 1 } { \int_use:N \c@iRow } { \int_use:N \c@jCol }
4048     }
4049   }
4050 }

4051 \cs_new_protected:Npn \l_@@_after_array:
4052 {

```

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`.

```

4053   \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
4054   \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.

```

4055   \bool_if:NT \g_@@_last_col_found_bool
4056   { \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`.

```

4057   \bool_if:NT \l_@@_last_col_without_value_bool
4058   { \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.

```

4059   \bool_if:NT \l_@@_last_row_without_value_bool
4060   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

4061   \tl_gput_right:Ne \g_@@_aux_tl
4062   {
4063     \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
4064     {
4065       \int_use:N \l_@@_first_row_int ,
4066       \int_use:N \c@iRow ,
4067       \int_use:N \g_@@_row_total_int ,
4068       \int_use:N \l_@@_first_col_int ,
4069       \int_use:N \c@jCol ,
4070       \int_use:N \g_@@_col_total_int

```

```

4071     }
4072   }
4073   \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`).

```

4074   \seq_if_empty:NF \g_@@_pos_of_blocks_seq
4075   {
4076     \tl_gput_right:Ne \g_@@_aux_tl
4077     {
4078       \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
4079       { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }
4080     }
4081   }
4082   \seq_if_empty:NF \g_@@_multicolumn_cells_seq
4083   {
4084     \tl_gput_right:Ne \g_@@_aux_tl
4085     {
4086       \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
4087       { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
4088       \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
4089       { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
4090     }
4091   }

```

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

```

4092   \@@_create_diag_nodes:

```

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

```

4093   \pgfpicture
4094   \@@_create_aliases_last:
4095   \str_if_empty:NF \l_@@_name_str \@@_create_alias_nodes:
4096   \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.

```

4097   \bool_if:NT \l_@@_parallelize_diags_bool
4098   {
4099     \int_gzero:N \g_@@_ddots_int
4100     \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.

```

4101     \dim_gzero:N \g_@@_delta_x_one_dim
4102     \dim_gzero:N \g_@@_delta_y_one_dim
4103     \dim_gzero:N \g_@@_delta_x_two_dim
4104     \dim_gzero:N \g_@@_delta_y_two_dim
4105   }
4106   \bool_set_false:N \l_@@_initial_open_bool
4107   \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.

```

4108   \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

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

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

```
4109 \@@_draw_dotted_lines:
```

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

```
4110 \clist_if_empty:NF \l_@@_corners_cells_clist
4111 {
4112   \bool_if:NTF \l_@@_no_cell_nodes_bool
4113   { \@@_error:n { corners~with~no~cell~nodes } }
4114   \@@_compute_corners:
4115 }
```

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`.

```
4116 \seq_concat:NNN \g_@@_pos_of_blocks_seq
4117 \g_@@_pos_of_blocks_seq
4118 \g_@@_future_pos_of_blocks_seq
4119 \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-*}`).

```
4120 \@@_adjust_pos_of_blocks_seq:
4121 \@@_deal_with_rounded_corners:
4122 \legacy_if:nF { measuring@ } \@@_draw_rules:
4123 \tl_gclear:N \g_@@_rules_tl
```

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

```
4124 \IfPackageLoadedT { tikz }
4125 {
4126   \tikzset
4127   {
4128     every~picture / .style =
4129     {
4130       overlay ,
4131       remember~picture ,
4132       name~prefix = \@@_env: -
4133     }
4134   }
4135 }
4136 \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
4137 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
4138 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
4139 \cs_set_eq:NN \OverBrace \@@_OverBrace
4140 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
4141 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
4142 \cs_set_eq:NN \line \@@_line
```

The LaTeX-style boolean `\ifmeasuring@` is used by `amsmath` during the phase of measure in environments such as `{align}`, etc.

```
4143 \legacy_if:nF { measuring@ } \g_@@_pre_code_after_tl
4144 \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.

```
4145 \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).

```
4146 \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`).

```

4147 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
4148 { \@@_rescan_for_spanish:N \g_nicematrix_code_after_tl }

4149 \bool_set_true:N \l_@@_in_code_after_bool
4150 \@@_security_font_commands:

```

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:`.

```

4151 \bool_set_true:N \l_@@_in_code_after_bool
4152 \if_mode_math:
4153   \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4154   \scan_stop:
4155 \else:
4156   $ % $
4157   \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4158   \scan_stop:
4159   $ % $
4160 \fi:
4161 \tl_gclear:N \g_nicematrix_code_after_tl
4162 \clist_if_empty:NF \g_@@_col_with_trees_clist \@@_draw_trees:
4163 \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.

```

4164 \seq_if_empty:NF \g_@@_rowlistcolors_seq \@@_clear_rowlistcolors_seq:
4165 \tl_if_empty:NF \g_@@_pre_code_before_tl
4166 {
4167   \tl_gput_right:Ne \g_@@_aux_tl
4168   {
4169     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
4170     { \exp_not:o \g_@@_pre_code_before_tl }
4171   }
4172   \tl_gclear:N \g_@@_pre_code_before_tl
4173 }
4174 \tl_if_empty:NF \g_nicematrix_code_before_tl
4175 {
4176   \tl_gput_right:Ne \g_@@_aux_tl
4177   {
4178     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
4179     { \exp_not:o \g_nicematrix_code_before_tl }
4180   }
4181   \tl_gclear:N \g_nicematrix_code_before_tl
4182 }

4183 \str_gclear:N \g_@@_name_env_str
4184 \@@_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`.

```

4185 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
4186 }

```

¹³e.g. `\color[rgb]{0.5,0.5,0}`


```

4187 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
4188 {
4189   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
4190   \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.

```

4191   \dim_set:Nn \l_@@_xdots_shorten_start_dim
4192     { 0.6 \l_@@_xdots_shorten_start_dim }
4193   \dim_set:Nn \l_@@_xdots_shorten_end_dim
4194     { 0.6 \l_@@_xdots_shorten_end_dim }
4195 }

```

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`.

```

4196 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4197 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

```

```

4198 \cs_new_protected:Npn \@@_create_alias_nodes:
4199 {
4200   \int_step_inline:nn \c@iRow
4201   {
4202     \pgfnodealias
4203       { \l_@@_name_str - ##1 - last }
4204       { \@@_env: - ##1 - \int_use:N \c@jCol }
4205   }
4206   \int_step_inline:nn \c@jCol
4207   {
4208     \pgfnodealias
4209       { \l_@@_name_str - last - ##1 }
4210       { \@@_env: - \int_use:N \c@iRow - ##1 }
4211   }
4212   \pgfnodealias
4213     { \l_@@_name_str - last - last }
4214     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
4215 }

```

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).

```

4216 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4217 {
4218   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4219     { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4220 }

```

The following command must *not* be protected.

```

4221 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4222 {
4223   { #1 }
4224   { #2 }
4225   {
4226     \int_compare:nNnTF { #3 } > { 98 }
4227     { \int_use:N \c@iRow }
4228     { #3 }
4229   }

```

```

4230 {
4231   \int_compare:nNnTF { #4 } > { 98 }
4232     { \int_use:N \c@jCol }
4233     { #4 }
4234 }
4235 { #5 }
4236 }

```

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).

```

4237 \AtBeginDocument
4238 {
4239   \cs_new_protected:Npe \@@_draw_dotted_lines:
4240     {
4241       \c_@@_pgfortikzpicture_tl
4242       \@@_draw_dotted_lines_i:
4243       \c_@@_endpgfortikzpicture_tl
4244     }
4245 }

```

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

```

4246 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4247 {
4248   \pgfrememberpicturepositiononpagetrue
4249   \pgf@relevantforpicturesizefalse
4250   \g_@@_HVdotsfor_lines_tl
4251   \g_@@_Vdots_lines_tl
4252   \g_@@_Ddots_lines_tl
4253   \g_@@_Iddots_lines_tl
4254   \g_@@_Cdots_lines_tl
4255   \g_@@_Ldots_lines_tl
4256 }

4257 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4258 {
4259   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4260   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4261 }

```

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

```

4262 \pgfdeclareshape { @@_diag_node }
4263 {
4264   \savedanchor { \five }
4265   {
4266     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4267     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4268   }
4269   \anchor { 5 } { \five }
4270   \anchor { center } { \pgfpointorigin }
4271   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4272   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4273   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4274   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4275   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4276   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4277   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4278   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4279   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4280   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4281 }

```

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).

```

4282 \cs_new_protected:Npn \@@_create_diag_nodes:
4283 {
4284   \pgfpicture
4285   \pgfrememberpicturepositiononpagetrue
4286   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4287   {
4288     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4289     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4290     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4291     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4292     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4293     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4294     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4295     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4296     \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.

```

4297     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4298     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4299     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4300     \str_if_empty:NF \l_@@_name_str
4301     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4302   }

```

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

```

4303   \int_set:Nn \l_tmpa_int { 1 + \int_max:nn \c@iRow \c@jCol } % modified
4304   \@@_qpoint:n { row - \int_min:nn \l_tmpa_int { \c@iRow + 1 } }
4305   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4306   \@@_qpoint:n { col - \int_min:nn \l_tmpa_int { \c@jCol + 1 } }
4307   \pgfcoordinate
4308   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4309   \pgfnodealias
4310   { \@@_env: - last }
4311   { \@@_env: - \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
4312   \str_if_empty:NF \l_@@_name_str
4313   {
4314     \pgfnodealias
4315     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4316     { \@@_env: - \int_use:N \l_tmpa_int }
4317     \pgfnodealias
4318     { \l_@@_name_str - last }
4319     { \@@_env: - last }
4320   }
4321   \endpgfpicture
4322 }

```

16 We draw the dotted lines

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

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

The command `\l_@@_find_extremities:nnnn` takes four arguments:

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

This command computes:

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

We provide first a version in the L3 syntax, and, then a version slightly more efficient.

```
\cs_new_protected:Npn \l_@@_find_extremities:nnnn #1 #2 #3 #4
{
  \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
  \int_set:Nn \l_@@_initial_i_int { #1 }
  \int_set:Nn \l_@@_initial_j_int { #2 }
  \int_set:Nn \l_@@_final_i_int { #1 }
  \int_set:Nn \l_@@_final_j_int { #2 }
  \bool_set_false:N \l_@@_stop_loop_bool
  \bool_do_until:Nn \l_@@_stop_loop_bool
  {
    \int_add:Nn \l_@@_final_i_int { #3 }
    \int_add:Nn \l_@@_final_j_int { #4 }
    \bool_set_false:N \l_@@_final_open_bool
    \int_compare:nNnTF { \l_@@_final_i_int } > { \l_@@_row_max_int }
    {
      \int_compare:nNnTF { #3 } = { 1 }
      { \bool_set_true:N \l_@@_final_open_bool }
      {
        \int_compare:nNnT { \l_@@_final_j_int } > { \l_@@_col_max_int }
        { \bool_set_true:N \l_@@_final_open_bool }
      }
    }
  }
  {
    \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
    {
      \int_compare:nNnT { #4 } = { -1 }
      { \bool_set_true:N \l_@@_final_open_bool }
    }
    {
      \int_compare:nNnT { \l_@@_final_j_int } > { \l_@@_col_max_int }
      {
        \int_compare:nNnT { #4 } = { 1 }
        { \bool_set_true:N \l_@@_final_open_bool }
      }
    }
  }
}
\bool_if:NTF \l_@@_final_open_bool
{
  \int_sub:Nn \l_@@_final_i_int { #3 }
  \int_sub:Nn \l_@@_final_j_int { #4 }
  \bool_set_true:N \l_@@_stop_loop_bool
}
```

```

{
  \cs_if_exist:cTF
  {
    @@ _ dotted _
    \int_use:N \l_@@_final_i_int -
    \int_use:N \l_@@_final_j_int
  }
  {
    \int_sub:Nn \l_@@_final_i_int { #3 }
    \int_sub:Nn \l_@@_final_j_int { #4 }
    \bool_set_true:N \l_@@_final_open_bool
    \bool_set_true:N \l_@@_stop_loop_bool
  }
  {
    \cs_if_exist:cTF
    {
      pgf @ sh @ ns @ \@@_env:
      - \int_use:N \l_@@_final_i_int
      - \int_use:N \l_@@_final_j_int
    }
    { \bool_set_true:N \l_@@_stop_loop_bool }
    {
      \cs_set_nopar:cpn
      {
        @@ _ dotted _
        \int_use:N \l_@@_final_i_int -
        \int_use:N \l_@@_final_j_int
      }
      { }
    }
  }
}
}
\bool_set_false:N \l_@@_stop_loop_bool
\int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
\bool_do_until:Nn \l_@@_stop_loop_bool
{
  \int_sub:Nn \l_@@_initial_i_int { #3 }
  \int_sub:Nn \l_@@_initial_j_int { #4 }
  \bool_set_false:N \l_@@_initial_open_bool
  \int_compare:nNnTF { \l_@@_initial_i_int } < { \l_@@_row_min_int }
  {
    \int_compare:nNnTF { #3 } = { 1 }
    { \bool_set_true:N \l_@@_initial_open_bool }
    {
      \int_compare:nNnT { \l_@@_initial_j_int } = { \l_tmpa_int }
      { \bool_set_true:N \l_@@_initial_open_bool }
    }
  }
}
{
  \int_compare:nNnTF { \l_@@_initial_j_int } < { \l_@@_col_min_int }
  {
    \int_compare:nNnT { #4 } = { 1 }
    { \bool_set_true:N \l_@@_initial_open_bool }
  }
  {
    \int_compare:nNnT { \l_@@_initial_j_int } > { \l_@@_col_max_int }
    {
      \int_compare:nNnT { #4 } = { -1 }
      { \bool_set_true:N \l_@@_initial_open_bool }
    }
  }
}
}

```

```

\bool_if:NTF \l_@@_initial_open_bool
{
  \int_add:Nn \l_@@_initial_i_int { #3 }
  \int_add:Nn \l_@@_initial_j_int { #4 }
  \bool_set_true:N \l_@@_stop_loop_bool
}
{
  \cs_if_exist:cTF
  {
    @@ _ dotted _
    \int_use:N \l_@@_initial_i_int -
    \int_use:N \l_@@_initial_j_int
  }
  {
    \int_add:Nn \l_@@_initial_i_int { #3 }
    \int_add:Nn \l_@@_initial_j_int { #4 }
    \bool_set_true:N \l_@@_initial_open_bool
    \bool_set_true:N \l_@@_stop_loop_bool
  }
  {
    \cs_if_exist:cTF
    {
      pgf @ sh @ ns @ \@@_env:
      - \int_use:N \l_@@_initial_i_int
      - \int_use:N \l_@@_initial_j_int
    }
    { \bool_set_true:N \l_@@_stop_loop_bool }
    {
      \cs_set_nopar:cpn
      {
        @@ _ dotted _
        \int_use:N \l_@@_initial_i_int -
        \int_use:N \l_@@_initial_j_int
      }
      { }
    }
  }
}
}
\seq_gput_right:Ne \g_@@_pos_of_xdots_seq
{
  { \int_use:N \l_@@_initial_i_int }
  { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
  { \int_use:N \l_@@_final_i_int }
  { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
  { }
}
}

```

The following version is slightly more efficient.

```

4323 \cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
4324 {

```

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

```

4325   \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4326   \l_@@_initial_i_int = #1
4327   \l_@@_initial_j_int = #2
4328   \l_@@_final_i_int = #1
4329   \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.

```

4330     \let \l_@@_stop_loop_bool \c_false_bool
4331     \bool_do_until:Nn \l_@@_stop_loop_bool
4332     {

```

We test if we are still in the matrix.

```

4333         \advance \l_@@_final_i_int by #3
4334         \advance \l_@@_final_j_int by #4
4335         \let \l_@@_final_open_bool \c_false_bool
4336         \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4337             \if_int_compare:w #3 = \c_one_int
4338                 \let \l_@@_final_open_bool \c_true_bool
4339             \else:
4340                 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4341                     \let \l_@@_final_open_bool \c_true_bool
4342                 \fi:
4343             \fi:
4344         \else:
4345             \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4346                 \if_int_compare:w #4 = -1
4347                     \let \l_@@_final_open_bool \c_true_bool
4348                 \fi:
4349             \else:
4350                 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4351                     \if_int_compare:w #4 = \c_one_int
4352                         \let \l_@@_final_open_bool \c_true_bool
4353                     \fi:
4354                 \fi:
4355             \fi:
4356         \fi:
4357         \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.

```

4358     {

```

We do a step backwards.

```

4359         \advance \l_@@_final_i_int by - #3
4360         \advance \l_@@_final_j_int by - #4
4361         \let \l_@@_stop_loop_bool \c_true_bool
4362     }

```

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`.

```

4363     {
4364         \cs_if_exist:cTF
4365         {
4366             @@ _ dotted _
4367             \int_use:N \l_@@_final_i_int -
4368             \int_use:N \l_@@_final_j_int
4369         }
4370         {
4371             \advance \l_@@_final_i_int by - #3
4372             \advance \l_@@_final_j_int by - #4
4373             \let \l_@@_final_open_bool \c_true_bool
4374             \let \l_@@_stop_loop_bool \c_true_bool
4375         }
4376         {
4377             \cs_if_exist:cTF
4378             {
4379                 pgf @ sh @ ns @ \@@_env:
4380                 - \int_use:N \l_@@_final_i_int
4381                 - \int_use:N \l_@@_final_j_int
4382             }

```

```
4383         { \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.

```
4384     {
4385         \cs_set_nopar:cpn
4386         {
4387             @@ _ dotted _
4388             \int_use:N \l_@@_final_i_int -
4389             \int_use:N \l_@@_final_j_int
4390         }
4391     { }
4392 }
4393 }
4394 }
4395 }
```

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

```
4396     \let \l_@@_stop_loop_bool \c_false_bool
```

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

```
4397     \l_tmpa_int = \l_@@_col_min_int
4398     \advance \l_tmpa_int by -1
4399     \bool_do_until:Nn \l_@@_stop_loop_bool
4400     {
4401         \advance \l_@@_initial_i_int by - #3
4402         \advance \l_@@_initial_j_int by - #4
4403         \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.

```
4404         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4405         \if_int_compare:w #3 = \c_one_int
4406             \let \l_@@_initial_open_bool \c_true_bool
4407         \else:
```

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

```
4408             \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4409                 \let \l_@@_initial_open_bool \c_true_bool
4410             \fi:
4411         \fi:
4412     \else:
4413         \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4414         \if_int_compare:w #4 = \c_one_int
4415             \let \l_@@_initial_open_bool \c_true_bool
4416         \fi:
4417     \else:
4418         \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4419         \if_int_compare:w #4 = -1
4420             \let \l_@@_initial_open_bool \c_true_bool
4421         \fi:
4422     \fi:
4423 \fi:
4424 \fi:
4425 \bool_if:NTF \l_@@_initial_open_bool
4426 {
4427     \advance \l_@@_initial_i_int by #3
4428     \advance \l_@@_initial_j_int by #4
```



```

4429     \let \l_@@_stop_loop_bool \c_true_bool
4430   }
4431   {
4432     \cs_if_exist:cTF
4433     {
4434       @@ _ dotted _
4435       \int_use:N \l_@@_initial_i_int -
4436       \int_use:N \l_@@_initial_j_int
4437     }
4438     {
4439       \advance \l_@@_initial_i_int by #3
4440       \advance \l_@@_initial_j_int by #4
4441       \let \l_@@_initial_open_bool \c_true_bool
4442       \let \l_@@_stop_loop_bool \c_true_bool
4443     }
4444     {
4445       \cs_if_exist:cTF
4446       {
4447         pgf @ sh @ ns @ \@@_env:
4448         - \int_use:N \l_@@_initial_i_int
4449         - \int_use:N \l_@@_initial_j_int
4450       }
4451       { \let \l_@@_stop_loop_bool \c_true_bool }
4452       {
4453         \cs_set_nopar:cpn
4454         {
4455           @@ _ dotted _
4456           \int_use:N \l_@@_initial_i_int -
4457           \int_use:N \l_@@_initial_j_int
4458         }
4459         { }
4460       }
4461     }
4462   }
4463 }

```

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.

```

4464   \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4465   {
4466     { \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`.

```

4467     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4468     { \int_use:N \l_@@_final_i_int }
4469     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4470     { }
4471   }
4472 }

```

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.

```

4473 \cs_new_protected:Npn \@@_open_shorten:
4474 {
4475   \bool_if:NT \l_@@_initial_open_bool
4476   { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4477   \bool_if:NT \l_@@_final_open_bool
4478   { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4479 }

```

The following command (*when it will be written*) will set the four counters `\l_@@_row_min_int`, `\l_@@_row_max_int`, `\l_@@_col_min_int` and `\l_@@_col_max_int` to the intersections of the submatrices which contains the cell of row `#1` and column `#2`. As of now, it's only the whole array (excepted exterior rows and columns).

```

4480 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4481 {
4482   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4483   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4484   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4485   \int_set_eq:NN \l_@@_col_max_int \c@jCol

```

If there is no submatrices, we will speed up a little for the potential other dotted lines to draw.

```

4486   \seq_if_empty:NTF \g_@@_submatrix_seq
4487   { \cs_set_eq:NN \@@_adjust_to_submatrix:nn \use_none:nn }
4488   {

```

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`.

```

4489     \seq_map_inline:Nn \g_@@_submatrix_seq
4490     { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4491   }
4492 }

```

`#1` and `#2` are the numbers of row and columns of the cell where the command of dotted line (ex.: `\Vdots`) has been issued. `#3`, `#4`, `#5` and `#6` are the specification (in i and j) of the submatrix we are analyzing.

Here is the programmation of that command with the the standard syntax of L3.

```

\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
{
  \bool_if:nT
  {
    \int_compare_p:n { #3 <= #1 <= #5 }
    &&
    \int_compare_p:n { #4 <= #2 <= #6 }
  }
  {
    \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
    \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
    \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
    \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
  }
}

```

However, for efficiency, we will use the following version.

```

4493 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4494 {
4495   \if_int_compare:w #3 > #1
4496   \else:
4497     \if_int_compare:w #1 > #5
4498     \else:
4499       \if_int_compare:w #4 > #2
4500       \else:
4501         \if_int_compare:w #2 > #6
4502         \else:
4503           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4504           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4505           \if_int_compare:w \l_@@_row_max_int > #5 \l_@@_row_max_int = #5 \fi:
4506           \if_int_compare:w \l_@@_col_max_int > #6 \l_@@_col_max_int = #6 \fi:
4507         \fi:
4508       \fi:
4509     \fi:
4510   \fi:
4511 }

```

```

4512 \cs_new_protected:Npn \@@_set_initial_coords:
4513 {
4514   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4515   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4516 }
4517 \cs_new_protected:Npn \@@_set_final_coords:
4518 {
4519   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4520   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4521 }
4522 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4523 {
4524   \pgfpointanchor
4525   {
4526     \@@_env:
4527     - \int_use:N \l_@@_initial_i_int
4528     - \int_use:N \l_@@_initial_j_int
4529   }
4530   { #1 }
4531   \@@_set_initial_coords:
4532 }
4533 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4534 {
4535   \pgfpointanchor
4536   {
4537     \@@_env:
4538     - \int_use:N \l_@@_final_i_int
4539     - \int_use:N \l_@@_final_j_int
4540   }
4541   { #1 }
4542   \@@_set_final_coords:
4543 }
4544 \cs_new_protected:Npn \@@_open_x_initial_dim:
4545 {
4546   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4547   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4548   {
4549     \cs_if_exist:cT
4550     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4551     {
4552       \pgfpointanchor
4553       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4554       { west }
4555       \dim_set:Nn \l_@@_x_initial_dim
4556       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4557     }
4558   }

```

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

```

4559   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4560   {
4561     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4562     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4563     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4564   }
4565 }
4566 \cs_new_protected:Npn \@@_open_x_final_dim:
4567 {
4568   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4569   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4570   {
4571     \cs_if_exist:cT
4572     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }

```

```

4573     {
4574         \pgfpointanchor
4575         { \l_@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4576         { east }
4577         \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4578         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4579     }
4580 }

```

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

```

4581     \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4582     {
4583         \l_@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4584         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4585         \dim_sub:Nn \l_@@_x_final_dim \col@sep
4586     }
4587 }

```

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.

```

4588 \cs_new_protected:Npn \l_@@_draw_Ldots:nnn #1 #2 #3
4589 {
4590     \l_@@_adjust_to_submatrix:nn { #1 } { #2 }
4591     \cs_if_free:cT { @ _ dotted _ #1 - #2 }
4592     {
4593         \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.

```

4594     \bool_if:NT \g_@@_aux_found_bool
4595     {
4596         {
4597             \l_@@_open_shorten:
4598             \int_if_zero:nTF { #1 }
4599             { \color { nicematrix-first-row } }
4600         }

```

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.

```

4601         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4602         { \color { nicematrix-last-row } }
4603     }
4604     \keys_set:nn { nicematrix / xdots } { #3 }
4605     \l_@@_color:o \l_@@_xdots_color_tl
4606     \l_@@_actually_draw_Ldots:
4607 }
4608 }
4609 }
4610 }

```

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

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

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

```

4611 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4612 {
4613   \bool_if:NTF \l_@@_initial_open_bool
4614   {
4615     \@@_open_x_initial_dim:
4616     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4617     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4618   }
4619   { \@@_set_initial_coords_from_anchor:n { base-east } }
4620   \bool_if:NTF \l_@@_final_open_bool
4621   {
4622     \@@_open_x_final_dim:
4623     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4624     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4625   }
4626   { \@@_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.

```

4627   \bool_lazy_all:nTF
4628   {
4629     \l_@@_initial_open_bool
4630     \l_@@_final_open_bool
4631     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4632   }
4633   {
4634     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4635     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4636   }

```

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 (?).

```

4637   {
4638     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4639     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4640   }
4641   \@@_draw_line:
4642 }

```

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.

```

4643 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4644 {
4645   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4646   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4647   {
4648     \@@_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.

```

4649   \bool_if:NT \g_@@_aux_found_bool
4650   {
4651     {
4652       \@@_open_shorten:
4653       \int_if_zero:nTF { #1 }
4654       { \color { nicematrix-first-row } }
4655     }

```

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.

```

4656   \int_compare:nNnT { #1 } = \l_@@_last_row_int

```

```

4657         { \color { nicematrix-last-row } }
4658     }
4659     \keys_set:nn { nicematrix / xdots } { #3 }
4660     \@@_color:o \l_@@_xdots_color_tl
4661     \@@_actually_draw_Cdots:
4662 }
4663 }
4664 }
4665 }

```

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.`

```

4666 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4667 {
4668     \bool_if:NTF \l_@@_initial_open_bool
4669         \@@_open_x_initial_dim:
4670         { \@@_set_initial_coords_from_anchor:n { mid-east } }
4671     \bool_if:NTF \l_@@_final_open_bool
4672         \@@_open_x_final_dim:
4673         { \@@_set_final_coords_from_anchor:n { mid-west } }
4674     \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4675     {
4676         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4677         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4678         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4679         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4680         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4681     }
4682     {
4683         \bool_if:NT \l_@@_initial_open_bool
4684         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4685         \bool_if:NT \l_@@_final_open_bool
4686         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4687     }
4688     \@@_draw_line:
4689 }
4690 \cs_new_protected:Npn \@@_open_y_initial_dim:
4691 {
4692     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4693     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4694     {
4695         \cs_if_exist:cT
4696         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4697         {
4698             \pgfpointanchor
4699             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4700             { north }
4701             \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4702             { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4703         }
4704     }
4705     \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4706     {

```

```

4707 \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4708 \dim_set:Nn \l_@@_y_initial_dim
4709 {
4710   \fp_to_dim:n
4711   {
4712     \pgf@y
4713     + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4714   }
4715 }
4716 }
4717 }
4718 \cs_new_protected:Npn \@@_open_y_final_dim:
4719 {
4720   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4721   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4722   {
4723     \cs_if_exist:cT
4724     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4725     {
4726       \pgfpointanchor
4727       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4728       { south }
4729       \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4730       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4731     }
4732   }
4733   \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4734   {
4735     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4736     \dim_set:Nn \l_@@_y_final_dim
4737     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4738   }
4739 }

```

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.

```

4740 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4741 {
4742   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4743   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4744   {
4745     \@@_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.

```

4746   \bool_if:NT \g_@@_aux_found_bool
4747   {
4748     {
4749       \@@_open_shorten:
4750       \int_if_zero:nTF { #2 }
4751       { \color { nicematrix-first-col } }
4752       {
4753         \int_compare:nNnT { #2 } = \l_@@_last_col_int
4754         { \color { nicematrix-last-col } }
4755       }
4756       \keys_set:nn { nicematrix / xdots } { #3 }
4757       \@@_color:o \l_@@_xdots_color_tl
4758       \bool_if:NTF \l_@@_Vbrace_bool
4759       \@@_actually_draw_Vbrace:
4760       \@@_actually_draw_Vdots:
4761     }
4762   }
4763 }
4764 }

```

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`.

```

4765 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4766 {
4767   \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4768   \@@_actually_draw_Vdots_i:
4769   \@@_actually_draw_Vdots_ii:
4770   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4771   \@@_draw_line:
4772 }

```

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

```

4773 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4774 {
4775   \@@_open_y_initial_dim:
4776   \@@_open_y_final_dim:
4777   \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4778 {
4779   \@@_qpoint:n { col - 1 }
4780   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4781   \dim_sub:Nn \l_@@_x_initial_dim
4782   { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4783 }
4784 {
4785   \bool_lazy_and:nnTF
4786   { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4787   { \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”.

```

4788 {
4789   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4790   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4791   \dim_add:Nn \l_@@_x_initial_dim
4792   { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4793 }

```

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

```

4794 {
4795   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4796   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4797   \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4798   \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4799 }
4800 }
4801 }

```

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

Now, the dotted line is *not* open on both sides (maybe open on only one side).
The main task is to determine the x -value of the dotted line to draw.

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

```

4802 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4803 {
4804   \bool_set_false:N \l_tmpa_bool
4805   \bool_if:NF \l_@@_initial_open_bool
4806   {
4807     \bool_if:NF \l_@@_final_open_bool
4808     {
4809       \@@_set_initial_coords_from_anchor:n { south-west }
4810       \@@_set_final_coords_from_anchor:n { north-west }
4811       \bool_set:Nn \l_tmpa_bool
4812         { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4813     }
4814   }

```

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

```

4815   \bool_if:NTF \l_@@_initial_open_bool
4816   {
4817     \@@_open_y_initial_dim:
4818     \@@_set_final_coords_from_anchor:n { north }
4819     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4820   }
4821   {
4822     \@@_set_initial_coords_from_anchor:n { south }
4823     \bool_if:NTF \l_@@_final_open_bool
4824     \@@_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.

```

4825     {
4826       \@@_set_final_coords_from_anchor:n { north }
4827       \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4828       {
4829         \dim_set:Nn \l_@@_x_initial_dim
4830         {
4831           \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4832             \l_@@_x_initial_dim \l_@@_x_final_dim
4833         }
4834       }
4835     }
4836   }
4837 }

```

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`.

```

4838 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4839 {
4840   \bool_if:NTF \l_@@_initial_open_bool
4841   \@@_open_y_initial_dim:
4842   { \@@_set_initial_coords_from_anchor:n { south } }
4843   \bool_if:NTF \l_@@_final_open_bool
4844   \@@_open_y_final_dim:
4845   { \@@_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.

```

4846 \int_if_zero:nTF \l_@@_initial_j_int
4847 {
4848   \@@_qpoint:n { col - 1 }
4849   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4850   \dim_sub:Nn \l_@@_x_initial_dim
4851   { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4852 }

```

Elsewhere, the brace must be drawn left flush.

```

4853 {
4854   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4855   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4856   \dim_add:Nn \l_@@_x_initial_dim
4857   { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4858 }

```

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

```

4859 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4860 \@@_draw_line:
4861 }

```

```

4862 \cs_new:Npn \@@_colsep:
4863 { \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.

```

4864 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4865 {
4866   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4867   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4868   {
4869     \@@_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.

```

4870   \bool_if:nTF \g_@@_aux_found_bool
4871   {
4872     {
4873       \@@_open_shorten:
4874       \keys_set:nn { nicematrix / xdots } { #3 }
4875       \@@_color:o \l_@@_xdots_color_tl
4876       \@@_actually_draw_Ddots:
4877     }
4878   }
4879 }
4880 }

```

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`.

```

4881 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4882 {
4883   \bool_if:NTF \l_@@_initial_open_bool
4884   {
4885     \@@_open_y_initial_dim:
4886     \@@_open_x_initial_dim:
4887   }
4888   { \@@_set_initial_coords_from_anchor:n { south-east } }
4889   \bool_if:NTF \l_@@_final_open_bool
4890   {
4891     \@@_open_x_final_dim:
4892     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4893   }
4894   { \@@_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.

```

4895   \bool_if:NT \l_@@_parallelize_diags_bool
4896   {
4897     \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).

```

4898     \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.

```

4899     {
4900       \dim_gset:Nn \g_@@_delta_x_one_dim
4901       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4902       \dim_gset:Nn \g_@@_delta_y_one_dim
4903       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4904     }

```

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`.

```

4905     {
4906       \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4907       {
4908         \dim_set:Nn \l_@@_y_final_dim
4909         {
4910           \l_@@_y_initial_dim +
4911           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4912           \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4913         }
4914       }
4915     }
4916   }
4917   \@@_draw_line:
4918 }

```

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.

```

4919 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4920 {
4921   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4922   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4923   {
4924     \@@_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.

```

4925     \bool_if:NT \g_@@_aux_found_bool
4926     {
4927     {
4928         \@@_open_shorten:
4929         \keys_set:nn { nicematrix / xdots } { #3 }
4930         \@@_color:o \l_@@_xdots_color_tl
4931         \@@_actually_draw_Iddots:
4932     }
4933     }
4934 }
4935 }

```

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`.

```

4936 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4937 {
4938     \bool_if:NTF \l_@@_initial_open_bool
4939     {
4940         \@@_open_y_initial_dim:
4941         \@@_open_x_initial_dim:
4942     }
4943     { \@@_set_initial_coords_from_anchor:n { south-west } }
4944     \bool_if:NTF \l_@@_final_open_bool
4945     {
4946         \@@_open_y_final_dim:
4947         \@@_open_x_final_dim:
4948     }
4949     { \@@_set_final_coords_from_anchor:n { north-east } }
4950     \bool_if:NT \l_@@_parallelize_diags_bool
4951     {
4952         \int_gincr:N \g_@@_iddots_int
4953         \int_compare:nNnTF \g_@@_iddots_int = 1
4954         {
4955             \dim_gset:Nn \g_@@_delta_x_two_dim
4956             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4957             \dim_gset:Nn \g_@@_delta_y_two_dim
4958             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4959         }
4960         {
4961             \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4962             {
4963                 \dim_set:Nn \l_@@_y_final_dim
4964                 {
4965                     \l_@@_y_initial_dim +
4966                     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4967                     \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4968                 }
4969             }
4970         }
4971     }
4972     \@@_draw_line:
4973 }

```

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`

```

4974 \cs_new_protected:Npn \@@_draw_line:
4975 {
4976   \pgfrememberpicturepositiononpagetrue
4977   \pgf@relevantforpicturesizefalse
4978   \bool_lazy_or:nnTF
4979     \l_@@_dotted_bool
4980     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4981     \@@_draw_standard_dotted_line:
4982     \@@_draw_unstandard_dotted_line:
4983 }

```

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.

```

4984 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4985 {
4986   \begin { scope }
4987     \@@_draw_unstandard_dotted_line:o
4988     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4989 }

```

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.

```

4990 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4991 {
4992   \@@_draw_unstandard_dotted_line:nooo
4993   { #1 }
4994   \l_@@_xdots_up_tl
4995   \l_@@_xdots_down_tl
4996   \l_@@_xdots_middle_tl
4997 }
4998 \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.

```

4999 \AtBeginDocument
5000 {
5001   \IfPackageLoadedT { tikz }
5002   {
5003     \tikzset
5004     {
5005       @@_node_above / .style = { sloped , above } ,
5006       @@_node_below / .style = { sloped , below } ,
5007       @@_node_middle / .style =
5008       {

```

```

5009         sloped ,
5010         inner~sep = \c_@@_innersep_middle_dim
5011     }
5012 }
5013 }
5014 }

5015 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
5016 {

```

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.

```

5017 \dim_zero_new:N \l_@@_l_dim
5018 \dim_set:Nn \l_@@_l_dim
5019 {
5020     \fp_to_dim:n
5021     {
5022         sqrt
5023         (
5024             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5025             +
5026             ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5027         )
5028     }
5029 }

```

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.

```

5030 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
5031 {
5032     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
5033     \@@_draw_unstandard_dotted_line_i:
5034 }

```

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

```

5035 \bool_if:NT \l_@@_xdots_h_labels_bool
5036 {
5037     \tikzset
5038     {
5039         @@_node_above / .style = { auto = left } ,
5040         @@_node_below / .style = { auto = right } ,
5041         @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
5042     }
5043 }
5044 \tl_if_empty:nF { #4 }
5045 { \tikzset { @@_node_middle / .append~style = { fill = white } } }

5046 \dim_zero:N \l_tmpa_dim
5047 \dim_zero:N \l_tmpb_dim
5048 \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
5049 {

```

We test whether the brace is vertical or horizontal.

```

5050 \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
5051 { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
5052 { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
5053 }
5054 {

```

```

5055 \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
5056 {
5057   \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
5058     { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
5059     { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
5060   }
5061 }
5062 \use:e
5063 {
5064   \exp_not:N \begin { scope }
5065     [ shift = {(\dim_use:N \l_tmpa_dim, \dim_use:N \l_tmpb_dim)} ]
5066   }
5067 \draw
5068 [ #1 ]
5069 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
5070 -- node [ @@_node_middle] { $ \scriptstyle #4 $ }
5071 node [ @@_node_below ] { $ \scriptstyle #3 $ }
5072 node [ @@_node_above ] { $ \scriptstyle #2 $ }
5073 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
5074 \end { scope }
5075 \end { scope }
5076 }
5077 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
5078 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
5079 {
5080   \dim_set:Nn \l_tmpa_dim
5081   {
5082     \l_@@_x_initial_dim
5083     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5084     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
5085   }
5086   \dim_set:Nn \l_tmpb_dim
5087   {
5088     \l_@@_y_initial_dim
5089     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5090     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
5091   }
5092   \dim_set:Nn \l_@@_tmpc_dim
5093   {
5094     \l_@@_x_final_dim
5095     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5096     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5097   }
5098   \dim_set:Nn \l_@@_tmpd_dim
5099   {
5100     \l_@@_y_final_dim
5101     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5102     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5103   }
5104   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
5105   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
5106   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
5107   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
5108 }

```

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

```

5109 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
5110 {
5111   {

```

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.

```

5112 \dim_zero_new:N \l_@@_l_dim
5113 \dim_set:Nn \l_@@_l_dim
5114 {
5115   \fp_to_dim:n
5116   {
5117     sqrt
5118     (
5119       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5120       +
5121       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5122     )
5123   }
5124 }

```

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.

```

5125 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
5126 {
5127   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
5128   \@@_draw_standard_dotted_line_i:
5129 }
5130 }
5131 \bool_lazy_all:nF
5132 {
5133   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
5134   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
5135   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
5136 }
5137 \@@_labels_standard_dotted_line:
5138 }
5139 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
5140 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
5141 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

5142 \int_set:Nn \l_tmpa_int
5143 {
5144   \dim_ratio:nn
5145   {
5146     \l_@@_l_dim
5147     - \l_@@_xdots_shorten_start_dim
5148     - \l_@@_xdots_shorten_end_dim
5149   }
5150   \l_@@_xdots_inter_dim
5151 }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

5152 \dim_set:Nn \l_tmpa_dim
5153 {
5154   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5155   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5156 }
5157 \dim_set:Nn \l_tmpb_dim
5158 {
5159   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5160   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5161 }

```


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.

```

5162 \dim_gadd:Nn \l_@@_x_initial_dim
5163 {
5164   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5165   \dim_ratio:nn
5166   {
5167     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5168     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5169   }
5170   { 2 \l_@@_l_dim }
5171 }
5172 \dim_gadd:Nn \l_@@_y_initial_dim
5173 {
5174   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5175   \dim_ratio:nn
5176   {
5177     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5178     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5179   }
5180   { 2 \l_@@_l_dim }
5181 }
5182 \pgf@relevantforpicturesizefalse
5183 \int_step_inline:nnn \c_zero_int \l_tmpa_int
5184 {
5185   \pgfpathcircle
5186   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5187   \l_@@_xdots_radius_dim
5188   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
5189   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
5190 }
5191 \pgfusepathqfill
5192 }

5193 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
5194 {
5195   \pgfscope
5196   \pgftransformshift
5197   {
5198     \pgfpointlineattime { 0.5 }
5199     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5200     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
5201   }
5202   \fp_set:Nn \l_tmpa_fp
5203   {
5204     atand
5205     (
5206       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
5207       \l_@@_x_final_dim - \l_@@_x_initial_dim
5208     )
5209   }
5210   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
5211   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
5212   \tl_if_empty:NF \l_@@_xdots_middle_tl
5213   {
5214     \begin { pgfscope }
5215     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
5216     \pgfnode
5217     { rectangle }
5218     { center }
5219     {
5220       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }

```

```

5221         {
5222             $ % $
5223             \scriptstyle \l_@@_xdots_middle_tl
5224             $ % $
5225         }
5226     }
5227     { }
5228     {
5229         \pgfsetfillcolor { white }
5230         \pgfusepath { fill }
5231     }
5232     \end { pgfscope }
5233 }
5234 \tl_if_empty:NF \l_@@_xdots_up_tl
5235 {
5236     \pgfnode
5237     { rectangle }
5238     { south }
5239     {
5240         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5241         {
5242             $ % $
5243             \scriptstyle \l_@@_xdots_up_tl
5244             $ % $
5245         }
5246     }
5247     { }
5248     { \pgfusepath { } }
5249 }
5250 \tl_if_empty:NF \l_@@_xdots_down_tl
5251 {
5252     \pgfnode
5253     { rectangle }
5254     { north }
5255     {
5256         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5257         {
5258             $ % $
5259             \scriptstyle \l_@@_xdots_down_tl
5260             $ % $
5261         }
5262     }
5263     { }
5264     { \pgfusepath { } }
5265 }
5266 \endpgfscope
5267 }

```

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.

```

5268 \AtBeginDocument
5269 {

```

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

```

5270 \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
5271 \cs_new_protected:Npn \@@_Ldots: { \@@_collect_options:n { \@@_Ldots_i } }
5272 \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5273 {
5274   \int_if_zero:nTF \c@jCol
5275   { \@@_error:nn { in~first~col } { \Ldots } }
5276   {
5277     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5278     { \@@_error:nn { in~last~col } { \Ldots } }
5279     {
5280       \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5281       { #1 , down = #2 , up = #3 , middle = #4 }
5282     }
5283   }
5284   \bool_if:NF \l_@@_nullify_dots_bool
5285   { \phantom { \ensuremath { \@@_old_ldots: } } }
5286   \bool_gset_true:N \g_@@_empty_cell_bool
5287 }

```

```

5288 \cs_new_protected:Npn \@@_Cdots: { \@@_collect_options:n { \@@_Cdots_i } }
5289 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5290 {
5291   \int_if_zero:nTF \c@jCol
5292   { \@@_error:nn { in~first~col } { \Cdots } }
5293   {
5294     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5295     { \@@_error:nn { in~last~col } { \Cdots } }
5296     {
5297       \@@_instruction_of_type:nnn { \c_false_bool } { \Cdots }
5298       { #1 , down = #2 , up = #3 , middle = #4 }
5299     }
5300   }
5301   \bool_if:NF \l_@@_nullify_dots_bool
5302   { \phantom { \ensuremath { \@@_old_cdots: } } }
5303   \bool_gset_true:N \g_@@_empty_cell_bool
5304 }

```

```

5305 \cs_new_protected:Npn \@@_Vdots: { \@@_collect_options:n { \@@_Vdots_i } }
5306 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5307 {
5308   \int_if_zero:nTF \c@iRow
5309   { \@@_error:nn { in~first~row } { \Vdots } }
5310   {
5311     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5312     { \@@_error:nn { in~last~row } { \Vdots } }
5313     {
5314       \@@_instruction_of_type:nnn { \c_false_bool } { \Vdots }
5315       { #1 , down = #2 , up = #3 , middle = #4 }
5316     }
5317   }
5318   \bool_if:NF \l_@@_nullify_dots_bool
5319   { \phantom { \ensuremath { \@@_old_vdots: } } }
5320   \bool_gset_true:N \g_@@_empty_cell_bool
5321 }

```

```

5322 \cs_new_protected:Npn \@@_Ddots: { \@@_collect_options:n { \@@_Ddots_i } }

```

```

5323 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5324 {
5325   \int_case:nnF \c@iRow
5326   {
5327     0 { \@@_error:nn { in-first~row } { \Ddots } }
5328     \l_@@_last_row_int { \@@_error:nn { in-last~row } { \Ddots } }
5329   }
5330   {
5331     \int_case:nnF \c@jCol
5332     {
5333       0 { \@@_error:nn { in-first~col } { \Ddots } }
5334       \l_@@_last_col_int { \@@_error:nn { in-last~col } { \Ddots } }
5335     }
5336     {
5337       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5338       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5339       { #1 , down = #2 , up = #3 , middle = #4 }
5340     }
5341   }
5342 }
5343 \bool_if:NF \l_@@_nullify_dots_bool
5344 { \phantom { \ensuremath { \@@_old_ddots: } } }
5345 \bool_gset_true:N \g_@@_empty_cell_bool
5346 }

5347 \cs_new_protected:Npn \@@_Iddots:
5348 { \@@_collect_options:n { \@@_Iddots_i } }
5349 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5350 {
5351   \int_case:nnF \c@iRow
5352   {
5353     0 { \@@_error:nn { in-first~row } { \Iddots } }
5354     \l_@@_last_row_int { \@@_error:nn { in-last~row } { \Iddots } }
5355   }
5356   {
5357     \int_case:nnF \c@jCol
5358     {
5359       0 { \@@_error:nn { in-first~col } { \Iddots } }
5360       \l_@@_last_col_int { \@@_error:nn { in-last~col } { \Iddots } }
5361     }
5362     {
5363       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5364       \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { Iddots }
5365       { #1 , down = #2 , up = #3 , middle = #4 }
5366     }
5367   }
5368   \bool_if:NF \l_@@_nullify_dots_bool
5369   { \phantom { \ensuremath { \@@_old_iddots: } } }
5370   \bool_gset_true:N \g_@@_empty_cell_bool
5371 }
5372 }

```

End of the \AddToHook.

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

```

5373 \keys_define:nn { nicematrix / Ddots }
5374 {
5375   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5376   draw-first .value_forbidden:n = true ,
5377 }

```

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

```

5378 \cs_new_protected:Npn \@@_Hspace:
5379 {
5380   \bool_gset_true:N \g_@@_empty_cell_bool
5381   \hspace
5382 }

```

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`.

```

5383 \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`.

```

5384 \cs_new:Npn \@@_Hdotsfor:
5385 {
5386   \bool_lazy_and:nnTF
5387   { \int_if_zero_p:n \c@jCol }
5388   { \int_if_zero_p:n \l_@@_first_col_int }
5389   {
5390     \bool_if:NTF \g_@@_after_col_zero_bool
5391     {
5392       \multicolumn { 1 } { c } { }
5393       \@@_Hdotsfor_i:
5394     }
5395     { \@@_fatal:n { Hdotsfor~in~col~0 } }
5396   }
5397   {
5398     \multicolumn { 1 } { c } { }
5399     \@@_Hdotsfor_i:
5400   }
5401 }

```

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:`).

```

5402 \AtBeginDocument
5403 {

```

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

```

5404   \cs_new_protected:Npn \@@_Hdotsfor_i:
5405   { \@@_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`).

```

5406   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m O { } E { _ ^ : } { { } { } { } } }
5407   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5408   {
5409     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5410     {
5411       \@@_Hdotsfor:nnnn
5412       { \int_use:N \c@iRow }
5413       { \int_use:N \c@jCol }
5414       { #2 }
5415       {
5416         #1 , #3 ,
5417         down = \exp_not:n { #4 } ,
5418         up = \exp_not:n { #5 } ,
5419         middle = \exp_not:n { #6 }
5420       }
5421     }
5422     \prg_replicate:nn { #2 - 1 }

```

```

5423     {
5424         &
5425         \multicolumn { 1 } { c } { }
5426         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5427     }
5428 }
5429 }

```

```

5430 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5431 {
5432     \bool_set_false:N \l_@@_initial_open_bool
5433     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5434     \int_set:Nn \l_@@_initial_i_int { #1 }
5435     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5436     \int_compare:nNnTF { #2 } = 1
5437     {
5438         \int_set:Nn \l_@@_initial_j_int 1
5439         \bool_set_true:N \l_@@_initial_open_bool
5440     }
5441     {
5442         \cs_if_exist:cTF
5443         {
5444             pgf @ sh @ ns @ \@@_env:
5445             - \int_use:N \l_@@_initial_i_int
5446             - \int_eval:n { #2 - 1 }
5447         }
5448         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5449         {
5450             \int_set:Nn \l_@@_initial_j_int { #2 }
5451             \bool_set_true:N \l_@@_initial_open_bool
5452         }
5453     }
5454     \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5455     {
5456         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5457         \bool_set_true:N \l_@@_final_open_bool
5458     }
5459     {
5460         \cs_if_exist:cTF
5461         {
5462             pgf @ sh @ ns @ \@@_env:
5463             - \int_use:N \l_@@_final_i_int
5464             - \int_eval:n { #2 + #3 }
5465         }
5466         { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5467         {
5468             \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5469             \bool_set_true:N \l_@@_final_open_bool
5470         }
5471     }
5472     \bool_if:NT \g_@@_aux_found_bool
5473     {
5474         {
5475             \@@_open_shorten:
5476             \int_if_zero:nTF { #1 }
5477             { \color { nicematrix-first-row } }
5478             {
5479                 \int_compare:nNnT { #1 } = \g_@@_row_total_int
5480                 { \color { nicematrix-last-row } }
5481             }

```

```

5482         \keys_set:nn { nicematrix / xdots } { #4 }
5483         \@@_color:o \l_@@_xdots_color_tl
5484         \@@_actually_draw_Ldots:
5485     }
5486 }

```

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).

```

5487     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5488     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - #1 } { } }
5489 }

```

```

5490 \AtBeginDocument
5491 {
5492     \cs_new_protected:Npn \@@_Vdotsfor:
5493     { \@@_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`).

```

5494     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m O { } E { _ ^ : } { { } { } { } } }
5495     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5496     {
5497         \bool_gset_true:N \g_@@_empty_cell_bool
5498         \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5499         {
5500             \@@_Vdotsfor:nnnn
5501             { \int_use:N \c@iRow }
5502             { \int_use:N \c@jCol }
5503             { #2 }
5504             {
5505                 #1 , #3 ,
5506                 down = \exp_not:n { #4 } ,
5507                 up = \exp_not:n { #5 } ,
5508                 middle = \exp_not:n { #6 }
5509             }
5510         }
5511     }
5512 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5513 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5514 {
5515     \bool_set_false:N \l_@@_initial_open_bool
5516     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

5517     \int_set:Nn \l_@@_initial_j_int { #2 }
5518     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5519     \int_compare:nNnTF { #1 } = 1
5520     {
5521         \int_set:Nn \l_@@_initial_i_int 1
5522         \bool_set_true:N \l_@@_initial_open_bool
5523     }
5524     {
5525         \cs_if_exist:cTF
5526         {
5527             pgf @ sh @ ns @ \@@_env:

```

```

5528         - \int_eval:n { #1 - 1 }
5529         - \int_use:N \l_@@_initial_j_int
5530     }
5531     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5532     {
5533         \int_set:Nn \l_@@_initial_i_int { #1 }
5534         \bool_set_true:N \l_@@_initial_open_bool
5535     }
5536 }
5537 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5538 {
5539     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5540     \bool_set_true:N \l_@@_final_open_bool
5541 }
5542 {
5543     \cs_if_exist:cTF
5544     {
5545         pgf @ sh @ ns @ \@@_env:
5546         - \int_eval:n { #1 + #3 }
5547         - \int_use:N \l_@@_final_j_int
5548     }
5549     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5550     {
5551         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5552         \bool_set_true:N \l_@@_final_open_bool
5553     }
5554 }
5555 \bool_if:NT \g_@@_aux_found_bool
5556 {
5557     {
5558         \@@_open_shorten:
5559         \int_if_zero:nTF { #2 }
5560         { \color { nicematrix-first-col } }
5561         {
5562             \int_compare:nNnT { #2 } = \g_@@_col_total_int
5563             { \color { nicematrix-last-col } }
5564         }
5565         \keys_set:nn { nicematrix / xdots } { #4 }
5566         \@@_color:o \l_@@_xdots_color_tl
5567         \bool_if:NTF \l_@@_Vbrace_bool
5568         \@@_actually_draw_Vbrace:
5569         \@@_actually_draw_Vdots:
5570     }
5571 }

```

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).

```

5572     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5573     { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5574 }

```

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

```

5575 \NewDocumentCommand \@@_rotate: { 0 { } }
5576 {
5577     \bool_gset_true:N \g_@@_rotate_bool
5578     \keys_set:nn { nicematrix / rotate } { #1 }
5579     \ignorespaces
5580 }

```


The command `\@@_rotate_p_col:` will be linked to `\rotate` in the the cells of the columns of type `p` and *al*.

```

5581 \cs_new_protected:Npn \@@_rotate_p_col: { \@@_error:n { rotate~in~p~col } }

5582 \keys_define:nn { nicematrix / rotate }
5583 {
5584   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5585   c .value_forbidden:n = true ,
5586   -90 .code:n = \bool_gset_true:N \g_@@_rotate_minus_bool ,
5587   -90 .value_forbidden:n = true ,
5588   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5589 }

```

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

```

5590 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5591 {
5592   \tl_if_empty:nTF { #2 }
5593     { #1 }
5594     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5595 }
5596 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5597 { \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).

```

5598 \AtBeginDocument
5599 {

```

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

```

5600   \tl_set_rescan:Nnn \l_tmpa_tl { }
5601   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5602   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5603   {
5604     {
5605       \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5606       \@@_color:o \l_@@_xdots_color_tl
5607       \use:e
5608       {
5609         \@@_line_i:nn
5610         { \@@_double_int_eval:n #2 - \q_stop }

```

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

```

5611         { \@@_double_int_eval:n #3 - \q_stop }
5612     }
5613 }
5614 }
5615 }
5616 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5617 {
5618     \bool_set_false:N \l_@@_initial_open_bool
5619     \bool_set_false:N \l_@@_final_open_bool
5620     \bool_lazy_or:nnTF
5621     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5622     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5623     { \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 } }

```

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

```

5624     { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5625 }
5626 \AtBeginDocument
5627 {
5628     \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5629     {

```

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:`.

```

5630         \c_@@_pgfortikzpicture_tl
5631         \@@_draw_line_iii:nn { #1 } { #2 }
5632         \c_@@_endpgfortikzpicture_tl
5633     }
5634 }

```

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

```

5635 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5636 {
5637     \pgfrememberpicturepositiononpagetrue
5638     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5639     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5640     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5641     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5642     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5643     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5644     \@@_draw_line:
5645 }

```

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.

```
5646 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNnT \c@iRow < }
5647 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNnF \c@jCol < }
```

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

```
5648 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5649 {
5650   \tl_gput_right:Ne \g_@@_row_style_tl
5651 }
```

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

```
5652   \exp_not:N
5653   \@@_if_row_less_than:nn
5654   { \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).

```
5655   {
5656     \exp_not:N
5657     \@@_if_col_greater_than:nn
5658     { \int_use:N \c@jCol }
5659     { \exp_not:n { #1 } \scan_stop: }
5660   }
5661 }
5662 }
5663 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
```

```
5664 \keys_define:nn { nicematrix / RowStyle }
5665 {
5666   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5667   cell-space-top-limit+ .code:n =
5668     \dim_set:Nn \l_tmpa_dim { \l_@@_cell_space_top_limit_dim + #1 } ,
5669   cell-space-top-limit+ .value_required:n = true ,
5670   cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
5671   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5672   cell-space-bottom-limit+ .code:n =
5673     \dim_set:Nn \l_tmpb_dim { \l_@@_cell_space_bottom_limit_dim + #1 } ,
5674   cell-space-bottom-limit+ .value_required:n = true ,
5675   cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
5676   cell-space-limits .meta:n =
5677     {
5678       cell-space-top-limit = #1 ,
5679       cell-space-bottom-limit = #1 ,
5680     } ,
5681   cell-space-limits+ .meta:n =
5682     {
5683       cell-space-top-limit += #1 ,
5684       cell-space-bottom-limit += #1 ,
5685     } ,
5686   cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
5687   color .tl_set:N = \l_@@_color_tl ,
5688   color .value_required:n = true ,
5689   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5690   nb-rows .code:n =
5691     \str_if_eq:eeTF { #1 } { * }
5692     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5693     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5694   nb-rows .value_required:n = true ,
5695   fill .tl_set:N = \l_@@_fill_tl ,
5696   fill .value_required:n = true ,
```

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

```

5697     opacity .tl_set:N = \l_@@_opacity_tl ,
5698     opacity .value_required:n = true ,
5699     rowcolor .tl_set:N = \l_@@_fill_tl ,
5700     rowcolor .value_required:n = true ,
5701     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5702     rounded-corners .default:n = 4 pt ,
5703     unknown .code:n =
5704       \@@_unknown_key:nn
5705       { nicematrix / RowStyle }
5706       { Unknown-key-for-RowStyle }
5707   }

```

```

5708 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5709 {
5710   \group_begin:
5711   \tl_clear:N \l_@@_fill_tl
5712   \tl_clear:N \l_@@_opacity_tl
5713   \tl_clear:N \l_@@_color_tl
5714   \int_set:Nn \l_@@_key_nb_rows_int 1
5715   \dim_zero:N \l_@@_rounded_corners_dim
5716   \dim_zero:N \l_tmpa_dim
5717   \dim_zero:N \l_tmpb_dim
5718   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5719   \tl_if_empty:NF \l_@@_fill_tl
5720   {
5721     \@@_add_opacity_to_fill:
5722     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5723     {

```

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

```

5724       \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5725       { \int_use:N \c@iRow - \int_use:N \c@jCol }
5726       {
5727         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5728         - *
5729       }
5730       { \dim_use:N \l_@@_rounded_corners_dim }
5731     }
5732   }
5733   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5734   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5735   {
5736     \@@_put_in_row_style:e
5737     {
5738       \@@_put_in_cell_after_hook:n
5739       {

```

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

```

5740       \dim_set:Nn \l_@@_cell_space_top_limit_dim
5741       { \dim_use:N \l_tmpa_dim }
5742     }
5743   }
5744 }

```

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

```

5745   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5746   {
5747     \@@_put_in_row_style:e
5748     {

```

```

5749         \@@_put_in_cell_after_hook:n
5750         {
5751             \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5752             { \dim_use:N \l_tmpb_dim }
5753         }
5754     }
5755 }

```

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

```

5756     \tl_if_empty:NF \l_@@_color_tl
5757     {
5758         \@@_put_in_row_style:e
5759         {
5760             \mode_leave_vertical:
5761             \@@_color:n { \l_@@_color_tl }
5762         }
5763     }

```

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

```

5764     \bool_if:NT \l_@@_bold_row_style_bool
5765     {
5766         \@@_put_in_row_style:n
5767         {
5768             \exp_not:n
5769             {
5770                 \if_mode_math:
5771                 $ % $
5772                 \bfseries \boldmath
5773                 $ % $
5774             \else:
5775                 \bfseries \boldmath
5776             \fi:
5777         }
5778     }
5779 }
5780 \group_end:
5781 \g_@@_row_style_tl
5782 \ignorespaces
5783 }

```

The following commande must *not* be protected.

```

5784 \cs_new:Npn \@@_rounded_from_row:n #1
5785 {
5786     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

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

```

5787     { \int_eval:n { #1 } - 1 }
5788     {
5789         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5790         - \exp_not:n { \int_use:N \c@jCol }
5791     }
5792     { \dim_use:N \l_@@_rounded_corners_dim }
5793 }

```

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).

```
5794 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5795 {
```

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.

```
5796 \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`.

```
5797 \str_if_in:nnF { #1 } { !! }
5798 {
5799 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

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

```
5800 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5801 }
5802 \int_if_zero:nTF \l_tmpa_int
```

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

```
5803 {
5804 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5805 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5806 }
```

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

```
5807 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5808 }
5809 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
```

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

```
5810 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5811 {
5812 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5813 {
```

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

```
5814 \group_begin:
5815 \pgfsetcornersarced
5816 { \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.

```
5817 \bool_if:NTF \l_@@_hvlines_bool
5818 {
```

```

5819     \pgfpathrectanglecorners
5820     {
5821         \pgfpointadd
5822         { \@@_qpoint:n { row-1 } }
5823         { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
5824     }
5825     {
5826         \pgfpointadd
5827         {
5828             \@@_qpoint:n
5829             { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
5830         }
5831         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5832     }
5833 }
5834 {
5835     \pgfpathrectanglecorners
5836     { \@@_qpoint:n { row-1 } }
5837     {
5838         \pgfpointadd
5839         {
5840             \@@_qpoint:n
5841             { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
5842         }
5843         { \pgfpoint \c_zero_dim \arrayrulewidth }
5844     }
5845 }
5846 \pgfusepath { clip }
5847 \group_end:

```

The TeX group was for \pgfsetcornersarced.

```

5848     }
5849 }

```

The command \@@_actually_color: will actually fill the rectangles, color by color, as specified in the sequence \g_@@_colors_seq.

```

5850 \cs_new_protected:Npn \@@_actually_color:
5851 {
5852     \pgfpicture
5853     \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.

```

5854     \@@_clip_with_rounded_corners:

```

We will now 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).

```

5855     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5856     {
5857         \int_compare:nNnTF { ##1 } = 1
5858         {
5859             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5860             \use:c { g_@@_color _ 1 _tl }
5861             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5862         }
5863         {
5864             \pgfscope
5865             \@@_color_opacity: ##2
5866             \use:c { g_@@_color _ ##1 _tl }
5867             \tl_gclear:c { g_@@_color _ ##1 _tl }
5868             \pgfusepath { fill }
5869             \endpgfscope
5870         }
5871     }

```

```

5872 \endpgfpicture
5873 }

```

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

```

5874 \cs_new_protected:Npn \@@_color_opacity:
5875 {
5876   \peek_meaning:NTF [
5877     \@@_color_opacity:w
5878     { \@@_color_opacity:w [ ] }
5879   }

```

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.

```

5880 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5881 {
5882   \tl_clear:N \l_tmpa_tl
5883   \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.

```

5884   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5885   \tl_if_empty:NTF \l_tmpb_tl
5886     \@declaredcolor
5887     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5888 }

```

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

```

5889 \keys_define:nn { nicematrix / color-opacity }
5890 {
5891   opacity .tl_set:N      = \l_tmpa_tl ,
5892   opacity .value_required:n = true
5893 }

```

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

```

5894 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5895 {
5896   \def \l_@@_rows_tl { #1 }
5897   \def \l_@@_cols_tl { #2 }
5898   \@@_cartesian_path:
5899 }

```

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

```

5900 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5901 {
5902   \tl_if_blank:nF { #2 }
5903   {
5904     \@@_add_to_colors_seq:en
5905     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5906     { \@@_cartesian_color:nn { #3 } { - } }
5907   }
5908 }

```

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

```

5909 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5910 {
5911   \tl_if_blank:nF { #2 }
5912   {
5913     \@@_add_to_colors_seq:en
5914     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }

```



```

5915         { \@@_cartesian_color:nn { - } { #3 } }
5916     }
5917 }

```

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

```

5918 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5919 {
5920     \tl_if_blank:nF { #2 }
5921     {
5922         \@@_add_to_colors_seq:en
5923         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5924         { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5925     }
5926 }

```

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

```

5927 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5928 {
5929     \tl_if_blank:nF { #2 }
5930     {
5931         \@@_add_to_colors_seq:en
5932         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5933         { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5934     }
5935 }

```

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

```

5936 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5937 {
5938     \@@_cut_on_hyphen:w #1 \q_stop
5939     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5940     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5941     \@@_cut_on_hyphen:w #2 \q_stop
5942     \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5943     \tl_set:Ne \l_@@_cols_tl { \l_@@_tmpd_tl - \l_tmpb_tl }

```

The command `\@@_cartesian_path:n` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

5944     \@@_cartesian_path:n { #3 }
5945 }

```

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

```

5946 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5947 {
5948     \clist_map_inline:nn { #3 }
5949     { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5950 }

```

```

5951 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5952 {
5953     \int_step_inline:nn \c@iRow
5954     {
5955         \int_step_inline:nn \c@jCol
5956         {
5957             \int_if_even:nTF { #####1 + ##1 }
5958             { \@@_cellcolor [ #1 ] { #2 } }
5959             { \@@_cellcolor [ #1 ] { #3 } }
5960             { ##1 - #####1 }
5961         }
5962     }
5963 }

```

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”.

```

5964 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5965 {
5966   \@@_rectanglecolor [ #1 ] { #2 }
5967   { 1 - 1 }
5968   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5969 }

5970 \keys_define:nn { nicematrix / rowcolors }
5971 {
5972   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5973   cols .tl_set:N = \l_@@_cols_tl ,
5974   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5975   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5976 }

```

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*.

```

5977 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5978 {

```

`\l_@@_colors_seq` will be the list of colors.

```

5979   \seq_clear_new:N \l_@@_colors_seq
5980   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5981   \tl_clear_new:N \l_@@_cols_tl
5982   \tl_set:Nn \l_@@_cols_tl { - }
5983   \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).

```

5984   \int_zero_new:N \l_@@_color_int
5985   \int_set:Nn \l_@@_color_int 1
5986   \bool_if:NT \l_@@_respect_blocks_bool
5987   {

```

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`).

```

5988     % modified 2026-02-18
5989     \seq_set_filter:Nnn \l_tmpa_seq \g_@@_pos_of_blocks_seq
5990     { \@@_not_in_exterior_p:nnnnn ##1 }
5991   }

```

#2 is the list of intervals of rows.

```

5992   \clist_map_inline:nn { #2 }
5993   {
5994     \tl_set:Nn \l_tmpa_tl { ##1 }
5995     \tl_if_in:NnTF \l_tmpa_tl { - }
5996     { \@@_cut_on_hyphen:w ##1 \q_stop }
5997     { \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.

```

5998     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5999     \int_set:Nn \l_@@_color_int
6000     { \bool_if:NNTF \l_@@_rowcolors_restart_bool { 1 } \l_tmpa_tl }
6001     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
6002     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
6003     {

```

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

```
6004 \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).

```
6005 \bool_if:NT \l_@@_respect_blocks_bool
6006 {
6007   \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
6008   { \@@_intersect_our_row_p:nnnnn ###1 }
6009   \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }
```

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

```
6010 }
6011 \tl_set:Ne \l_@@_rows_tl
6012 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
```

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

```
6013 \tl_set:Ne \l_@@_color_tl
6014 {
6015   \@@_color_index:n
6016   {
6017     \int_mod:nn
6018     { \l_@@_color_int - 1 }
6019     { \seq_count:N \l_@@_colors_seq }
6020     + 1
6021   }
6022 }
6023 \tl_if_empty:NF \l_@@_color_tl
6024 {
6025   \use:e
6026   {
6027     \@@_add_to_colors_seq:nn
6028     { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
6029     { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
6030   }
6031 }
6032 \int_incr:N \l_@@_color_int
6033 \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
6034 }
6035 }
6036 }
```

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.

```
6037 \cs_new:Npn \@@_color_index:n #1
6038 {
```

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

```
6039 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
6040 { \@@_color_index:n { #1 - 1 } }
6041 { \seq_item:Nn \l_@@_colors_seq { #1 } }
6042 }
```

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

```
6043 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
6044 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }
```

The braces around `#3` and `#4` are mandatory.

```
6045 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
6046 {
6047   \int_compare:nNnT { #3 } > \l_tmpb_int
6048   { \int_set:Nn \l_tmpb_int { #3 } }
6049 }
```

```

6050 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
6051 {
6052   \int_if_zero:nTF { #4 }
6053   \prg_return_false:
6054   {
6055     \int_compare:nNnTF { #2 } > \c@jCol
6056     \prg_return_false:
6057     \prg_return_true:
6058   }
6059 }

```

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

```

6060 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
6061 {
6062   \int_compare:nNnTF { #1 } > \l_tmpa_int
6063   \prg_return_false:
6064   {
6065     \int_compare:nNnTF \l_tmpa_int > { #3 }
6066     \prg_return_false:
6067     \prg_return_true:
6068   }
6069 }

```

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).

```

6070 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
6071 {
6072   \dim_compare:nNnTF { #1 } = \c_zero_dim
6073   {
6074     \bool_if:NTF \l_@@_nocolor_used_bool
6075     { \@@_cartesian_path_normal_ii: }
6076     {
6077       \clist_if_empty:NTF \l_@@_corners_cells_clist
6078       { \@@_cartesian_path_normal_i:n { #1 } }
6079       { \@@_cartesian_path_normal_ii: }
6080     }
6081   }
6082   { \@@_cartesian_path_normal_i:n { #1 } }
6083 }

```

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.

```

6084 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
6085 {
6086   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

6087   \clist_map_inline:Nn \l_@@_cols_tl
6088   {

```

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

```

6089     \def \l_tmpa_tl { ##1 }
6090     \tl_if_in:NnTF \l_tmpa_tl { - }
6091     { \@@_cut_on_hyphen:w ##1 \q_stop }
6092     { \def \l_tmpb_tl { ##1 } }
6093     \tl_if_empty:NTF \l_tmpa_tl
6094     { \def \l_tmpa_tl { 1 } }
6095     {

```

```

6096     \str_if_eq:eeT \l_tmpa_tl { * }
6097     { \def \l_tmpa_tl { 1 } }
6098 }
6099 \int_compare:nNnT \l_tmpa_tl > \g_@@_col_total_int
6100 { \@@_error:n { Invalid~col~number } }
6101 \tl_if_empty:NTF \l_tmpb_tl
6102 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6103 {
6104     \str_if_eq:eeT \l_tmpb_tl { * }
6105     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6106 }
6107 \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
6108 { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

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

```

6109 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6110 \@@_qpoint:n { col - \l_tmpa_tl }
6111 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
6112 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6113 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6114 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
6115 \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.

```

6116 \clist_map_inline:Nn \l_@@_rows_tl
6117 {
6118     \def \l_tmpa_tl { #####1 }
6119     \tl_if_in:NnTF \l_tmpa_tl { - }
6120     { \@@_cut_on_hyphen:w #####1 \q_stop }
6121     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
6122     \tl_if_empty:NTF \l_tmpa_tl
6123     { \def \l_tmpa_tl { 1 } }
6124     {
6125         \str_if_eq:eeT \l_tmpa_tl { * }
6126         { \def \l_tmpa_tl { 1 } }
6127     }
6128     \tl_if_empty:NTF \l_tmpb_tl
6129     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6130     {
6131         \str_if_eq:eeT \l_tmpb_tl { * }
6132         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6133     }
6134     \int_compare:nNnT \l_tmpa_tl > \g_@@_row_total_int
6135     { \@@_error:n { Invalid~row~number } }
6136     \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
6137     { \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`.

```

6138 \cs_if_exist:cF
6139 { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
6140 {
6141     \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
6142     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6143     \@@_qpoint:n { row - \l_tmpa_tl }
6144     \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6145     \pgfpathrectanglecorners
6146     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6147     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6148 }
6149 }
6150 }
6151 }

```

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

```

6152 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
6153 {
6154   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6155   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6156   \clist_map_inline:Nn \l_@@_cols_tl
6157   {
6158     \@@_qpoint:n { col - ##1 }
6159     \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
6160     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6161     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6162     \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
6163     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

6164     \clist_map_inline:Nn \l_@@_rows_tl
6165     {
6166       \@@_if_in_corner:nF { #####1 - ##1 }
6167       {
6168         \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
6169         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6170         \@@_qpoint:n { row - #####1 }
6171         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6172         \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
6173         {
6174           \pgfpathrectanglecorners
6175           { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6176           { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6177         }
6178       }
6179     }
6180   }
6181 }

```

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`).

```

6182 \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

```

6183 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
6184 {
6185   \bool_set_true:N \l_@@_nocolor_used_bool
6186   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6187   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6188   \clist_map_inline:Nn \l_@@_rows_tl
6189   {
6190     \clist_map_inline:Nn \l_@@_cols_tl
6191     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
6192   }
6193 }

```

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.

```

6194 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
6195 {
6196   \clist_set_eq:NN \l_tmpa_clist #1

```

```

6197 \clist_clear:N #1
6198 \clist_map_inline:Nn \l_tmpa_clist
6199 {

```

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

```

6200 \def \l_tmpa_tl { ##1 }
6201 \tl_if_in:NnTF \l_tmpa_tl { - }
6202 { \@@_cut_on_hyphen:w ##1 \q_stop }
6203 { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
6204 \bool_lazy_or:nnT
6205 { \str_if_eq_p:ee \l_tmpa_tl { * } }
6206 { \tl_if_blank_p:o \l_tmpa_tl }
6207 { \def \l_tmpa_tl { 1 } }
6208 \bool_lazy_or:nnT
6209 { \str_if_eq_p:ee \l_tmpb_tl { * } }
6210 { \tl_if_blank_p:o \l_tmpb_tl }
6211 { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6212 \int_compare:nNnT \l_tmpb_tl > { #2 }
6213 { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6214 \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
6215 { \clist_put_right:Nn #1 { #####1 } }
6216 }
6217 }

```

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

```

6218 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
6219 {
6220 \tl_gput_right:Ne \g_@@_pre_code_before_tl
6221 {

```

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`).

```

6222 \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
6223 { \int_use:N \c@iRow - \int_use:N \c@jCol }
6224 }
6225 \ignorespaces
6226 }

6227 \NewDocumentCommand \@@_cellcolor_error { 0 { } m }
6228 { \@@_error:n { cellcolor~in~Block } }
6229 % \end{macrocode}
6230 %
6231 % \begin{macrocode}
6232 \NewDocumentCommand \@@_rowcolor_error { 0 { } m }
6233 { \@@_error:n { rowcolor~in~Block } }
6234 % \end{macrocode}
6235 %
6236 % \bigskip
6237 % The following command will be linked to |\rowcolor| in the tabular.
6238 % \begin{macrocode}
6239 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
6240 {
6241 \tl_gput_right:Ne \g_@@_pre_code_before_tl
6242 {
6243 \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
6244 { \int_use:N \c@iRow - \int_use:N \c@jCol }
6245 { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
6246 }
6247 \ignorespaces
6248 }

```

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

```

6249 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
6250 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around #2 and #3 are mandatory.

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

```

6251 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
6252 {

```

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`.

```

6253 \seq_gclear:N \g_tmpa_seq
6254 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6255 { \@@_rowlistcolors_tabular:nnnn #1 }
6256 \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`.

```

6257 \seq_gput_right:Ne \g_@@_rowlistcolors_seq
6258 {
6259 { \int_use:N \c@iRow }
6260 { \exp_not:n { #1 } }
6261 { \exp_not:n { #2 } }
6262 { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
6263 }
6264 \ignorespaces
6265 }

```

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`).

```

6266 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
6267 {
6268 \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).

```

6269 { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
6270 {
6271 \tl_gput_right:Ne \g_@@_pre_code_before_tl
6272 {
6273 \@@_rowlistcolors
6274 [ \exp_not:n { #2 } ]
6275 { #1 - \int_eval:n { \c@iRow - 1 } }
6276 { \exp_not:n { #3 } }
6277 [ \exp_not:n { #4 } ]
6278 }
6279 }
6280 }

```

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.

```

6281 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:

```



```

6282 {
6283   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6284     { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
6285   \seq_gclear:N \g_@@_rowlistcolors_seq
6286 }

6287 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6288 {
6289   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6290     { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6291 }

```

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.

```

6292 \NewDocumentCommand \@@_columnncolor_preamble { 0 { } m }
6293 {

```

With the following line, we test whether the cell is the first one that we actually encounter in its column (don't forget that some rows may be incomplete and that an instruction `\multicolumn` may be used, leading to cells which are not "evaluated").

```

6294   \seq_if_in:NVF \g_@@_columnncolor_seq \c@jCol
6295   {
6296     \seq_gput_right:NV \g_@@_columnncolor_seq \c@jCol

```

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).

```

6297     \tl_gput_left:Ne \g_@@_pre_code_before_tl
6298     {
6299       \exp_not:N \columnncolor [ #1 ]
6300       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6301     }
6302   }
6303 }

```

```

6304 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6305 {
6306   \clist_map_inline:nn { #1 }
6307   {
6308     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6309       { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6310     \columnncolor { nocolor } { ##1 }
6311   }
6312 }

6313 \cs_new_protected:Npn \@@_EmptyRow:n #1
6314 {
6315   \clist_map_inline:nn { #1 }
6316   {
6317     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6318       { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6319     \rowcolor { nocolor } { ##1 }
6320   }
6321 }

```

The following command must *not* be protected.

```

6322 \cs_new:Npn \@@_FalseRow
6323 {
6324   \noalign
6325   {

```

```

6326     \skip_vertical:n
6327     {
6328         - \fp_to_dim:n { \normalbaselineskip * \arraystretch }
6329         + \l_@@_width_of_false_dim
6330     }
6331 }
6332 \rowcolor { nocolor }
6333 \tl_gput_right:Ne \g_nicematrix_code_before_tl
6334 { \exp_not:N \EmptyRow { \arabic { iRow } } }
6335 \\\
6336 }
6337 \cs_new:Npn \@@_FalseRow_in_cell: { \@@_error:n { Misuse-of-FalseRow } }

```

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`.

```

6338 \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`.

```

6339 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6340 {
6341     \int_if_zero:nTF \l_@@_first_col_int
6342     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6343     {
6344         \int_if_zero:nTF \c@jCol
6345         {
6346             \int_compare:nNnF \c@iRow = { -1 }
6347             {
6348                 \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 }
6349                 { #1 }
6350             }
6351         }
6352         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6353     }
6354 }

```

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.

```

6355 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6356 {
6357     \int_if_zero:nF \c@iRow
6358     {
6359         \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6360         {
6361             \int_compare:nNnT \c@jCol > \c_zero_int

```

```

6362         { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6363     }
6364 }
6365 }

```

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:nNtT \c@iRow < \l_@@_last_row_int`).

The following command will be used for `\Toprule`, `\Bottomrule` and `\Midrule`.

```

6366 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6367 {
6368     \IfPackageLoadedTF { tikz }
6369     {
6370         \IfPackageLoadedTF { booktabs }
6371         { #2 }
6372         { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6373     }
6374     { \@@_error:nn { TopRule~without~tikz } { #1 } }
6375 }
6376 \NewExpandableDocumentCommand { \@@_TopRule } { }
6377 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
6378 \cs_new:Npn \@@_TopRule_i:
6379 {
6380     \noalign \bgroup
6381     \peek_meaning:NTF [
6382     { \@@_TopRule_ii: }
6383     { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6384 }
6385 \NewDocumentCommand \@@_TopRule_ii: { o }
6386 {
6387     \tl_gput_right:Ne \g_@@_rules_tl
6388     {
6389         \@@_draw_hrulerule:n
6390         {
6391             position = \int_eval:n { \c@iRow + 1 } ,
6392             tikz =
6393             {
6394                 line-width = #1 ,
6395                 yshift = 0.25 \arrayrulewidth ,
6396                 shorten-< = - 0.5 \arrayrulewidth
6397             } ,
6398             total-width = #1
6399         }
6400     }
6401     \skip_vertical:n { \belowrulesep + #1 }
6402     \egroup
6403 }
6404 \NewExpandableDocumentCommand { \@@_BottomRule } { }
6405 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }
6406 \cs_new:Npn \@@_BottomRule_i:
6407 {
6408     \noalign \bgroup
6409     \peek_meaning:NTF [
6410     { \@@_BottomRule_ii: }
6411     { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6412 }
6413 \NewDocumentCommand \@@_BottomRule_ii: { o }
6414 {
6415     \tl_gput_right:Ne \g_@@_rules_tl
6416     {
6417         \@@_draw_hrulerule:n

```

```

6418         {
6419             position = \int_eval:n { \c@iRow + 1 } ,
6420             tikz =
6421             {
6422                 line-width = #1 ,
6423                 yshift = 0.25 \arrayrulewidth ,
6424                 shorten-< = - 0.5 \arrayrulewidth
6425             } ,
6426             total-width = #1 ,
6427         }
6428     }
6429     \skip_vertical:N \aboverulesep
6430     \@@_create_row_node_i:
6431     \skip_vertical:n { #1 }
6432     \egroup
6433 }
6434 \NewExpandableDocumentCommand { \@@_MidRule } { }
6435 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6436 \cs_new:Npn \@@_MidRule_i:
6437 {
6438     \noalign \bgroup
6439     \peek_meaning:NTF [
6440         { \@@_MidRule_ii: }
6441         { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6442     ]
6443 \NewDocumentCommand \@@_MidRule_ii: { o }
6444 {
6445     \skip_vertical:N \aboverulesep
6446     \@@_create_row_node_i:
6447     \tl_gput_right:Ne \g_@@_rules_tl
6448     {
6449         \@@_draw_hrulerule:n
6450         {
6451             position = \int_eval:n { \c@iRow + 1 } ,
6452             tikz =
6453             {
6454                 line-width = #1 ,
6455                 yshift = 0.25 \arrayrulewidth ,
6456                 shorten-< = - 0.5 \arrayrulewidth
6457             } ,
6458             total-width = #1 ,
6459         }
6460     }
6461     \skip_vertical:n { \belowrulesep + #1 }
6462     \egroup
6463 }

```

General system for drawing rules

```

6464 \AtBeginDocument
6465 {
6466     \cs_new_protected:Npe \@@_draw_rules:
6467     {
6468         \c_@@_pgfortikzpicture_tl
6469         \@@_draw_rules_i:
6470         \c_@@_endpgfortikzpicture_tl
6471     }
6472 }
6473 \cs_new_protected:Npn \@@_draw_rules_i:
6474 {
6475     \bool_lazy_all:nT

```

```

6476 {
6477   { \clist_if_empty_p:N \l_@@_corners_clist }
6478   { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
6479   { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
6480   { \seq_if_empty_p:N \g_@@_pos_of_stroken_blocks_seq }
6481   { ! \l_@@_fix_vertex_bool }
6482 }
6483 {
6484   \socket_assign_plug:nn { nicematrix / draw-hrule } { quick }
6485   \socket_assign_plug:nn { nicematrix / draw-vrule } { quick }
6486 }
6487 \pgfrememberpicturepositiononpagetrue
6488 \pgf@relevantforpicturesizefalse
6489 \pgfsetrectcap
6490 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6491 \CT@arc@
6492 \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_key_hlines:
6493 \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_key_vlines:
6494 \g_@@_rules_tl
6495 }

```

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_hrule: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”.

```

6496 \keys_define:nn { nicematrix / rules-after }
6497 {
6498   position .int_set:N = \l_@@_position_int ,
6499   start .int_set:N = \l_@@_start_int ,
6500   start .initial:n = 1 ,
6501   end .int_set:N = \l_@@_end_int ,
6502   multiplicity .code:n = \@@_set_multiplicity:n { #1 } ,
6503   dotted .code:n =
6504     \bool_set_true:N \l_@@_dotted_bool
6505     \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
6506     \socket_assign_plug:nn { nicematrix / hsegment } { dotted } ,
6507   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]`.

```

6508   color .code:n =
6509     \@@_set_CTarc:n { #1 }
6510     \CT@arc@
6511     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6512   color .value_required:n = true ,
6513   sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6514   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
  }
}

6515 total-width .dim_set:N = \l_@@_rule_width_after_dim ,
6516 unknown .code:n =

```

```

6517 \@@_unknown_key:nn
6518 { nicematrix / rules-after }
6519 { Unknown-key-for-a-rule } ,
6520 tikz .value_required:n = true
6521 }

```

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.

```

6522 \AtBeginDocument
6523 {
6524   \IfPackageLoadedTF { tikz }
6525   {
6526     \keys_define:nn { nicematrix / rules-after }
6527     {
6528       tikz .code:n =
6529         \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 }
6530         \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
6531         \socket_assign_plug:nn { nicematrix / hsegment } { tikz } ,
6532       dashed .meta:n =
6533       {
6534         tikz = nicematrix / dashed ,
6535         total-width = \pgflinewidth
6536       }
6537     }
6538   }
6539   {
6540     \keys_define:nn { nicematrix / rules-after }
6541     { tikz .code:n = \@@_error:n { tikz~without~tikz } }
6542   }
6543 }

6544 \cs_new_protected:Npn \@@_set_multiplicity:n #1
6545 {
6546   \int_set:Nn \l_@@_multiplicity_int { #1 }
6547   \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
6548   \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
6549 }

6550 \AtBeginDocument
6551 {
6552   \IfPackageLoadedT { tikz }
6553   {
6554     \tikzset
6555     {
6556       nicematrix / dashed / .style =
6557       {
6558         dash-pattern = on~4~pt~off~2pt ,
6559         line-cap = butt
6560       }
6561     }
6562     \cs_if_exist:cT { tikz@library@decorations@loaded }
6563     { \tikzset { nicematrix / dashed / .append-style = dash-expand-off } }
6564   }
6565 }

```

The vertical rules

`\@@_draw_vrule:n` and the socket `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.

```

6566 \cs_new_protected:Npn \@@_draw_vrule:n #1
6567 {

```

The group is for the options.

```

6568 {
6569   \int_set_eq:NN \l_@@_end_int \c@iRow
6570   \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).

```

6571   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6572     { \socket_use:n { nicematrix / draw-vrule } }
6573 }
6574 }

```

The socket “`nicematrix / draw-vrule`” will draw a vertical rule. That vertical rule may potentially be composed of several disconnected segments.

The plug `general` will break the vertical rule in several segments.

The plug `quick` will draw directly the vertical rule. That plug is used when we are sure that the whole rule must be drawn, that is to say when:

- there is no corners;
- there is no blocks;
- there is no implicit blocks created by the continuous dotted lines;
- there is no stroken blocks.

```

6575 \socket_new:nn { nicematrix / draw-vrule } { 0 }
6576 \socket_new_plug:nnn { nicematrix / draw-vrule } { general }
6577 {
6578   \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`.

```

6579   \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).

```

6580   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6581   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6582   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6583     \l_tmpa_tl
6584   {

```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```

6585     \@@_test_v:
6586     \bool_if:NTF \g_tmpa_bool
6587     {
6588       \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.

```

6589       { \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl }
6590       { \socket_use:nn { nicematrix / draw-at-odd-vertex-v } }
6591     }
6592     {
6593       \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6594       {
6595         \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }

```

The socket `vsegment` is defined below with its four plugs.

```

6596       \socket_use:n { nicematrix / vsegment }
6597       \int_zero:N \l_@@_segment_start_int
6598     }
6599   }
6600 }
6601 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int

```

```

6602     {
6603         \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6604         \socket_use:n { nicematrix / vsegment }
6605     }
6606     \group_end:
6607 }
6608 \socket_assign_plug:nn { nicematrix / draw-vrule } { general }
6609 \socket_new_plug:nnn { nicematrix / draw-vrule } { quick }
6610 {
6611     \group_begin:
6612     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6613     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6614     \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
6615     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6616     \socket_use:n { nicematrix / vsegment }
6617     \group_end:
6618 }

```

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.

```

6619 \cs_new_protected:Npn \@@_test_v:
6620 {
6621     \bool_gset_true:N \g_tmpa_bool
6622     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6623     \bool_if:NT \g_tmpa_bool
6624     {
6625         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6626         { \@@_test_vline_in_block:nnnnn ##1 }
6627         \bool_if:NT \g_tmpa_bool
6628         {
6629             \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6630             { \@@_test_vline_in_block:nnnnn ##1 }
6631             \bool_if:NT \g_tmpa_bool
6632             {
6633                 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6634                 { \@@_test_vline_in_stroken_block:nnnn ##1 }
6635             }
6636         }
6637     }
6638 }
6639 \socket_new:nn { nicematrix / draw-at-odd-vertex-v } { 0 }
6640 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-v } { active }
6641 {
6642     {
6643         \int_compare:nNnTF \l_@@_position_int > \c@jCol
6644         { \bool_set_false:N \l_tmpa_bool }
6645         {
6646             \@@_test_h:
6647             \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6648         }
6649         \int_compare:nNnTF \l_@@_position_int = 1
6650         { \bool_gset_false:N \g_tmpa_bool }
6651         {
6652             \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl - 1 } }
6653             \@@_test_h:
6654         }
6655         \bool_gset:Nn \g_tmpa_bool
6656         { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6657     }
6658     \bool_if:NT \g_tmpa_bool

```



```

6659     {
6660       \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }
6661       \socket_use:n { nicematrix / vsegment }
6662       \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl
6663     }
6664   }

6665 \cs_new_protected:Npn \@@_test_in_corner_v:
6666 {
6667   \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6668   {
6669     \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6670     { \bool_set_false:N \g_tmpa_bool }
6671   }
6672   {
6673     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6674     {
6675       \int_compare:nNnTF \l_tmpb_tl = 1
6676       { \bool_set_false:N \g_tmpa_bool }
6677       {
6678         \@@_if_in_corner:nT
6679         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6680         { \bool_set_false:N \g_tmpa_bool }
6681       }
6682     }
6683   }
6684 }

```

For the drawing of the (vertical) segments, we will use a socket with four plugs :

- a plug `standard` 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, including the key `dashed` of `custom-line`.

```

6685 \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`.

```

6686 \socket_new_plug:nnn { nicematrix / vsegment } { standard }
6687 {
6688   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6689   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6690   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6691   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6692   \pgfusepathqstroke
6693 }

6694 \socket_assign_plug:nn { nicematrix / vsegment } { standard }

6695 \socket_new_plug:nnn { nicematrix / vsegment } { multiple }
6696 {
6697   \dim_add:Nn \l_@@_x_initial_dim
6698   {
6699     - 0.5 \l_@@_rule_width_after_dim
6700     +
6701     ( \arrayrulewidth * \l_@@_multiplicity_int
6702       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6703   }
6704   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6705   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y

```

```

6706 \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6707 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6708 \bool_lazy_and:nnT
6709 { \cs_if_exist_p:N \CT@drsc@ }
6710 { ! \tl_if_blank_p:o \CT@drsc@ }
6711 {
6712   {
6713     \CT@drsc@
6714     \dim_add:Nn \l_@@_y_initial_dim { 0.5 \arrayrulewidth }
6715     \dim_sub:Nn \l_@@_y_final_dim { 0.5 \arrayrulewidth }
6716     \dim_set:Nn \l_@@_tmpd_dim
6717     {
6718       \l_@@_x_initial_dim - ( \doublerulesep + \arrayrulewidth )
6719       * ( \l_@@_multiplicity_int - 1 )
6720     }
6721     \pgfpathrectanglecorners
6722     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6723     { \pgfpoint \l_@@_tmpd_dim \l_@@_y_final_dim }
6724     \pgfusepath { fill }
6725   }
6726 }
6727 \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6728 \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6729 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6730 {
6731   \dim_sub:Nn \l_@@_x_initial_dim { \arrayrulewidth + \doublerulesep }
6732   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6733   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6734 }
6735 \pgfusepathqstroke
6736 }

6737 \socket_new_plug:nnn { nicematrix / vsegment } { dotted }
6738 {
6739   \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6740   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6741   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6742   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6743   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6744   \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`

```

6745 \@@_draw_line:
6746 }

```

```

6747 \socket_new_plug:nnn { nicematrix / vsegment } { tikz }
6748 {
6749   {

```

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).

```

6750 \tl_if_empty:NF \l_@@_rule_color_tl
6751 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6752 \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6753 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6754 \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6755 \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }

```

The following line can't be short-cutted.

```

6756 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6757 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6758 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim ) --
6759 ( \l_@@_x_initial_dim , \l_@@_y_final_dim ) ;
6760 }
6761 }

```

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).

```

6762 \cs_new_protected:Npn \@@_draw_key_vlines:
6763 {
6764 {
6765 \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
6766 \int_set:Nn \l_@@_end_int \c@iRow

```

There is a currying in the following code.

```

6767 \str_if_eq:eeTF \l_@@_vlines_clist { all }
6768 { \@@_lines_step_inline:n \c@jCol }
6769 { \clist_map_inline:Nn \l_@@_vlines_clist }
6770 {
6771 \int_set:Nn \l_@@_position_int { ##1 }
6772 \socket_use:n { nicematrix / draw-vrule }
6773 }
6774 }
6775 }

```

The following command is used in `\@@_draw_key_vlines:` and in `\@@_draw_key_hlines:`.

```

6776 \cs_new_protected:Npn \@@_lines_step_inline:n #1
6777 {
6778 \int_step_inline:nnn
6779 { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6780 {
6781 \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6782 { #1 }
6783 { \int_eval:n { #1 + 1 } }
6784 }
6785 }

```

The horizontal rules

`\@@_draw_hrule:n` and the socket `draw-hrule` 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.

```

6786 \cs_new_protected:Npn \@@_draw_hrule:n #1
6787 {
6788 {
6789 \int_set_eq:NN \l_@@_end_int \c@jCol
6790 \keys_set_known:nn { nicematrix / rules-after } { #1 }

```

```

6791     \socket_use:nn { nicematrix / draw-hrule }
6792   }
6793 }

```

The socket “`nicematrix / draw-hrule`” will draw an horizontal rule. That horizontal rule may potentially be composed of several disconnected segments.

The plug `general` will break the vertical rule in several segments.

The plug `quick` will draw directly the vertical rule. That plug is used when we are sure that the whole rule must be drawn, that is to say when:

- there is no corners;
- there is no blocks;
- there is no implicit blocks created by the continuous dotted lines;
- there is no stroken blocks.

```

6794 \socket_new:nn { nicematrix / draw-hrule } { 0 }
6795 \socket_new_plug:nnn { nicematrix / draw-hrule } { general }
6796 {
6797   \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`.

```

6798   \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).

```

6799   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6800   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6801   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6802     \l_tmpb_tl
6803   {

```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```

6804     \@@_test_h:
6805     \bool_if:NTF \g_tmpa_bool
6806     {
6807       \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.

```

6808       { \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl }
6809       { \socket_use:n { nicematrix / draw-at-odd-vertex-h } }
6810     }
6811   {
6812     \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6813     {
6814       \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }

```

The socket `hsegment` is defined below with its four plugs.

```

6815     \socket_use:n { nicematrix / hsegment }
6816     \int_zero:N \l_@@_segment_start_int
6817   }
6818 }
6819 }
6820 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6821 {
6822   \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6823   \socket_use:n { nicematrix / hsegment }
6824 }
6825 \group_end:
6826 }

```

```

6827 \socket_assign_plug:nn { nicematrix / draw-hrule } { general }
6828 \socket_new_plug:nnn { nicematrix / draw-hrule } { quick }
6829 {
6830   \group_begin:
6831   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6832   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6833   \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
6834   \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6835   \socket_use:n { nicematrix / hsegment }
6836   \group_end:
6837 }

```

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.

```

6838 \cs_new_protected:Npn \@@_test_h:
6839 {
6840   \bool_gset_true:N \g_tmpa_bool
6841   \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6842   \bool_if:NT \g_tmpa_bool
6843   {
6844     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6845     { \@@_test_hline_in_block:nnnnn ##1 }
6846     \bool_if:NT \g_tmpa_bool
6847     {
6848       \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6849       { \@@_test_hline_in_block:nnnnn ##1 }
6850       \bool_if:NT \g_tmpa_bool
6851       {
6852         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6853         { \@@_test_hline_in_stroken_block:nnnn ##1 }
6854       }
6855     }
6856   }
6857 }

6858 \socket_new:nn { nicematrix / draw-at-odd-vertex-h } { 0 }
6859 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-h } { active }
6860 {
6861   {
6862     \int_compare:nNnTF \l_@@_position_int > \c@iRow
6863     { \bool_set_false:N \l_tmpa_bool }
6864     {
6865       \@@_test_v:
6866       \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6867     }
6868     \int_compare:nNnTF \l_@@_position_int = 1
6869     { \bool_gset_false:N \g_tmpa_bool }
6870     {
6871       \tl_set:N \l_tmpa_tl { \int_eval:n { \l_tmpa_tl - 1 } }
6872       \@@_test_v:
6873     }
6874     \bool_gset:Nn \g_tmpa_bool
6875     { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6876   }
6877   \bool_if:NT \g_tmpa_bool
6878   {
6879     \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }
6880     \socket_use:n { nicematrix / hsegment }
6881     \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl
6882   }
6883 }

```

```

6884 \cs_new_protected:Npn \@@_test_in_corner_h:
6885 {
6886   \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6887   {
6888     \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6889     { \bool_set_false:N \g_tmpa_bool }
6890   }
6891   {
6892     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6893     {
6894       \int_compare:nNnTF \l_tmpa_tl = 1
6895       { \bool_set_false:N \g_tmpa_bool }
6896       {
6897         \@@_if_in_corner:nT
6898         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6899         { \bool_set_false:N \g_tmpa_bool }
6900       }
6901     }
6902   }
6903 }

```

For the drawing of the (horizontal) segments, we will use a socket with four plugs :

- a plug `standard` for 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, including the key `dashed` of `custom-line`.

```

6904 \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`.

```

6905 \socket_new_plug:nnn { nicematrix / hsegment } { standard }
6906 {
6907   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6908   \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6909   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6910   \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6911   \pgfusepathqstroke
6912 }
6913 \socket_assign_plug:nn { nicematrix / hsegment } { standard }
6914 \socket_new_plug:nnn { nicematrix / hsegment } { multiple }
6915 {
6916   \dim_add:Nn \l_@@_y_initial_dim
6917   {
6918     - 0.5 \l_@@_rule_width_after_dim
6919     +
6920     ( \arrayrulewidth * \l_@@_multiplicity_int
6921       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6922   }
6923   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6924   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6925   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6926   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6927   \bool_lazy_and:nnT
6928   { \cs_if_exist_p:N \CT@drsc@ }
6929   { ! \tl_if_blank_p:o \CT@drsc@ }
6930   {
6931     {

```

```

6932 \CT@drsc@
6933 \dim_set:Nn \l_@@_tmpd_dim
6934 {
6935     \l_@@_y_initial_dim - ( \doublerulesep + \arrayrulewidth )
6936     * ( \l_@@_multiplicity_int - 1 )
6937 }
6938 \pgfpathrectanglecorners
6939 { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6940 { \pgfpoint \l_@@_x_final_dim \l_@@_tmpd_dim }
6941 \pgfusepathqfill
6942 }
6943 }
6944 \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6945 \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
6946 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6947 {
6948     \dim_sub:Nn \l_@@_y_initial_dim { \arrayrulewidth + \doublerulesep }
6949     \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6950     \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
6951 }
6952 \pgfusepathqstroke
6953 }

```

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}
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

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}
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6954 \socket_new_plugin:nnn { nicematrix / hsegment } { dotted }
6955 {
6956     \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6957     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6958     \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6959     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6960     \int_compare:nNnT \l_@@_segment_start_int = 1
6961     {
6962         \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6963         \bool_if:NF \g_@@_delims_bool
6964         { \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.

```

6965     \tl_if_eq:NnF \g_@@_left_delim_tl (
6966     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6967     )
6968     \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6969     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6970     \int_compare:nNnT \l_@@_segment_end_int = \c@jCol
6971     {
6972         \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6973         \bool_if:NF \g_@@_delims_bool

```

```

6974         { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6975         \tl_if_eq:NnF \g_@@_right_delim_tl )
6976         { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6977     }

```

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`

```

6978     \@@_draw_line:
6979 }

```

```

6980 \socket_new_plug:nnn { nicematrix / hsegment } { tikz }
6981 {
6982     {

```

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).

```

6983     \tl_if_empty:NF \l_@@_rule_color_tl
6984     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6985     \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6986     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6987     \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6988     \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6989     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6990     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6991     ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
6992     -- ( \l_@@_x_final_dim , \l_@@_y_initial_dim ) ;
6993 }
6994 }

```

The command `\@@_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).

```

6995 \cs_new_protected:Npn \@@_draw_key_hlines:
6996 {
6997     {
6998         \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
6999         \int_set:Nn \l_@@_end_int \c@jCol

```

There is a currying in the following code.

```

7000     \str_if_eq:eeTF \l_@@_hlines_clist { all }
7001     { \@@_lines_step_inline:n \c@iRow }
7002     { \clist_map_inline:Nn \l_@@_hlines_clist }
7003     {
7004         \int_set:Nn \l_@@_position_int { ##1 }
7005         \socket_use:n { nicematrix / draw-hrule }
7006     }
7007 }
7008 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

7009 \cs_set:Npn \@@_Hline: { \noalign \bgroup \@@_Hline_i:n { 1 } }

```


The argument of the command `\@@_Hline_i:n` is the number of successive `\Hline` found.

```

7010 \cs_set:Npn \@@_Hline_i:n #1
7011 {
7012   \peek_remove_spaces:n
7013   {
7014     \peek_meaning:NTF \Hline
7015     { \@@_Hline_ii:nn { #1 + 1 } }
7016     { \@@_Hline_iii:n { #1 } }
7017   }
7018 }
7019 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
7020 \cs_set:Npn \@@_Hline_iii:n #1
7021 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
7022 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
7023 {
7024   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
7025   \skip_vertical:N \l_@@_rule_width_before_dim
7026   \tl_gput_right:Ne \g_@@_rules_tl
7027   {

```

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

```

\@@_draw_hrule: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:

```

7028 {
7029   \int_compare:nNnT { #1 } > 1 { \@@_set_multiplicity:n { #1 } }
7030   \int_set:Nn \l_@@_position_int { \int_eval:n { \c@iRow + 1 } }
7031   \dim_set:Nn \l_@@_rule_width_after_dim
7032   { \dim_use:N \l_@@_rule_width_before_dim }
7033   \@@_draw_hrule:n { #2 }
7034 }
7035 }
7036 \egroup
7037 }

```

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`).

```

7038 \cs_new_protected:Npn \@@_custom_line:n #1
7039 {
7040   \str_clear:N \l_@@_letter_str
7041   \str_clear:N \l_@@_command_str
7042   \str_clear:N \l_@@_ccommand_str
7043   \tl_clear_new:N \l_@@_other_keys_tl
7044   \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
7045   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
7046   {
7047     \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }

```

We delete the last character which is a space.

```

7048     \str_set:Ne \l_@@_command_str
7049     { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
7050   }
7051   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
7052   {
7053     \str_set:Ne \l_@@_ccommand_str
7054     { \str_tail:N \l_@@_ccommand_str }
7055     \str_set:Ne \l_@@_ccommand_str
7056     { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
7057   }

```

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.

```

7058   \bool_lazy_all:nTF
7059   {
7060     { \str_if_empty_p:N \l_@@_letter_str }
7061     { \str_if_empty_p:N \l_@@_command_str }
7062     { \str_if_empty_p:N \l_@@_ccommand_str }
7063   }
7064   { \@@_error:n { No~letter~and~no~command } }
7065   { \@@_custom_line_i:o \l_@@_other_keys_tl }
7066 }
7067 \keys_define:nn { nicematrix / custom-line }
7068 {
7069   letter .str_set:N = \l_@@_letter_str ,
7070   letter .value_required:n = true ,
7071   command .str_set:N = \l_@@_command_str ,
7072   command .value_required:n = true ,
7073   ccommand .str_set:N = \l_@@_ccommand_str ,
7074   ccommand .value_required:n = true
7075 }
7076 \cs_new_protected:Npn \@@_custom_line_i:n #1
7077 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

7078   \bool_set_false:N \l_@@_tikz_rule_bool
7079   \bool_set_false:N \l_@@_dotted_rule_bool
7080   \bool_set_false:N \l_@@_color_bool
7081   \keys_set:nn { nicematrix / custom-line-bis } { #1 }
7082   \bool_if:NT \l_@@_tikz_rule_bool
7083   {
7084     \IfPackageLoadedF { tikz }
7085     { \@@_error:n { tikz~in~custom~line~without~tikz } }
7086     \bool_if:NT \l_@@_color_bool
7087     { \@@_error:n { color~in~custom~line~with~tikz } }
7088   }
7089   \bool_if:NT \l_@@_dotted_rule_bool
7090   {
7091     \int_compare:nNnT \l_@@_multiplicity_int > 1
7092     { \@@_error:n { key~multiplicity~with~dotted } }
7093   }
7094   \str_if_empty:NF \l_@@_letter_str
7095   {
7096     \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
7097     { \@@_error:n { Several~letters } }
7098     {
7099       \tl_if_in:NoTF

```

```

7100         \c_@@_forbidden_letters_str
7101         \l_@@_letter_str
7102         { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
7103     {

```

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.

```

7104         \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
7105         { \@@_v_custom_line:nn { #1 } }
7106     }
7107 }
7108 }
7109 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
7110 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
7111 }
7112 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
7113 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
7114 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{nicematrix/rules-after}`). That's why the following set of keys has some keys which are no-op.

```

7115 \keys_define:nn { nicematrix / custom-line-bis }
7116 {
7117     multiplicity .int_set:N = \l_@@_multiplicity_int ,
7118     color .code:n = \bool_set_true:N \l_@@_color_bool ,
7119     color .value_required:n = true ,
7120     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
7121     tikz .value_required:n = true ,
7122     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7123     dotted .value_forbidden:n = true ,
7124     dashed .meta:n =
7125     {
7126         tikz = nicematrix / dashed ,
7127         total-width = \pgflinewidth
7128     } ,
7129     total-width .code:n = { } ,
7130     total-width .value_required:n = true ,
7131     sep-color .code:n = { } ,
7132     sep-color .value_required:n = true ,
7133     unknown .code:n =
7134     \@@_unknown_key:nn
7135     { nicematrix / custom-line-bis }
7136     { Unknown-key-for~custom-line }
7137 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

7138 \bool_new:N \l_@@_dotted_rule_bool
7139 \bool_new:N \l_@@_tikz_rule_bool
7140 \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).

```

7141 \keys_define:nn { nicematrix / custom-line-width }
7142 {
7143     multiplicity .int_set:N = \l_@@_tmpc_int ,
7144     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
7145     total-width .code:n = \dim_set:Nn \l_@@_rule_width_before_dim { #1 }
7146     \bool_set_true:N \l_@@_total_width_bool ,

```

```

7147     total-width .value_required:n = true ,
7148     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7149 }

```

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_hruler:n (which is in the internal \CodeAfter).

```

7150 \cs_new_protected:Npn \@@_h_custom_line:n #1
7151 {

```

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).

```

7152     \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
7153     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
7154 }

```

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_hruler:n (which is in the internal \CodeAfter).

```

7155 \cs_new_protected:Npn \@@_c_custom_line:n #1
7156 {

```

Here, we need an expandable command since it begins with an \noalign.

```

7157     \exp_args:Nc \DeclareExpandableDocumentCommand
7158     { nicematrix - \l_@@_ccommand_str }
7159     { 0 { } m }
7160     {
7161         \noalign
7162         {
7163             \@@_compute_rule_width:n { #1 , ##1 }
7164             \skip_vertical:n \l_@@_rule_width_before_dim
7165             \clist_map_inline:nn
7166             { ##2 }
7167             { \@@_c_custom_line_i:nn { #1 , ##1 } { ####1 } }
7168         }
7169     }
7170     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
7171 }

```

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*.

```

7172 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
7173 {
7174     \tl_if_in:nnTF { #2 } { - }
7175     { \@@_cut_on_hyphen:w #2 \q_stop }
7176     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
7177     \tl_gput_right:Ne \g_@@_rules_tl
7178     {
7179         \@@_draw_hruler:n
7180         {
7181             #1 ,
7182             start = \l_tmpa_tl ,
7183             end = \l_tmpb_tl ,
7184             position = \int_eval:n { \c@iRow + 1 } ,
7185             total-width = \dim_use:N \l_@@_rule_width_before_dim
7186         }
7187     }
7188 }

```

```

7189 \cs_new_protected:Npn \@@_compute_rule_width:n #1
7190 {
7191   \bool_set_false:N \l_@@_tikz_rule_bool
7192   \bool_set_false:N \l_@@_total_width_bool
7193   \bool_set_false:N \l_@@_dotted_rule_bool
7194   \keys_set_known:n { nicematrix / custom-line-width } { #1 }
7195   \bool_if:NF \l_@@_total_width_bool
7196   {
7197     \bool_if:NTF \l_@@_dotted_rule_bool
7198     { \dim_set:Nn \l_@@_rule_width_before_dim { 2 \l_@@_xdots_radius_dim } }
7199     {
7200       \bool_if:NF \l_@@_tikz_rule_bool
7201       {
7202         \dim_set:Nn \l_@@_rule_width_before_dim
7203         {
7204           \arrayrulewidth * \l_@@_tmpc_int
7205           + \doublerulesep * ( \l_@@_tmpc_int - 1 )
7206         }
7207       }
7208     }
7209   }
7210 }

```

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

```

7211 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
7212 {
7213   \str_if_eq:nnTF { #2 } { [ ] }
7214   { \@@_v_custom_line_i:nw { #1 } [ ] }
7215   { \@@_v_custom_line_ii:nn { #2 } { #1 } }
7216 }
7217 \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
7218 { \@@_v_custom_line:nn { #1 , #2 } }
7219 \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
7220 {
7221   \@@_compute_rule_width:n { #2 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

7222 \tl_gput_right:Ne \g_@@_array_preamble_tl
7223 {
7224   \exp_not:N !
7225   { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_before_dim } }
7226 }
7227 \tl_gput_right:Ne \g_@@_rules_tl
7228 {

```

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.

```

7229 {
7230   \dim_set:Nn \l_@@_rule_width_after_dim
7231   { \dim_use:N \l_@@_rule_width_before_dim }
7232   \int_set:Nn \l_@@_position_int { \int_eval:n { \c@jCol + 1 } }
7233   \@@_draw_vrule:n { #2 }
7234 }
7235 }
7236 \@@_rec_preamble:n #1
7237 }

```

```

7238 \@@_custom_line:n
7239 { letter = : , command = \hdottedline , ccommand = \cdottedline, dotted }

```

The key default-line

```

7240 \keys_define:nn { nicematrix / default-line }
7241 {
7242   multiplicity .int_set:N = \l_@@_multiplicity_int ,
7243   color .code:n = \bool_set_true:N \l_@@_color_bool ,
7244   color .value_required:n = true ,
7245   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7246   dotted .value_forbidden:n = true ,
7247   total-width .code:n = { } ,
7248   total-width .value_required:n = true ,
7249   sep-color .code:n = { } ,
7250   sep-color .value_required:n = true ,
7251   unknown .code:n =
7252     \@@_unknown_key:nn
7253       { nicematrix / default-line }
7254       { Unknown-key-for~default-line }
7255 }
7256 \AddToHook{ package / tikz / after }
7257 {
7258   \keys_define:nn { nicematrix / default-line }
7259   {
7260     tikz .code:n =
7261       \bool_set_true:N \l_@@_tikz_rule_bool
7262       \tl_set:Nn \l_@@_tikz_rule_tl { #1 } ,
7263     tikz .value_required:n = true ,
7264     dashed .meta:n =
7265       {
7266         tikz = nicematrix / dashed ,
7267         total-width = \pgflinewidth
7268       }
7269   }
7270   \@@_custom_line:n
7271   {
7272     letter = ; ,
7273     command = \hdashedline ,
7274     ccommand = \cdashedline ,
7275     tikz = nicematrix / dashed ,
7276     total-width = \pgflinewidth
7277   }
7278 }
7279 \cs_new_protected:Npn \@@_default_line:n #1
7280 {
7281   \bool_set_false:N \l_@@_tikz_rule_bool
7282   \bool_set_false:N \l_@@_dotted_rule_bool
7283   \bool_set_false:N \l_@@_color_bool
7284   \keys_set:nn { nicematrix / default-line } { #1 }
7285   \bool_if:NT \l_@@_tikz_rule_bool
7286   {
7287     \IfPackageLoadedF { tikz }
7288     { \@@_error:n { tikz~in~custom~line~without~tikz } }
7289     \bool_if:NT \l_@@_color_bool
7290     { \@@_error:n { color~in~custom~line~with~tikz } }
7291   }
7292   \bool_if:NT \l_@@_dotted_rule_bool
7293   {
7294     \int_compare:nNnT \l_@@_multiplicity_int > 1
7295     { \@@_error:n { key~multiplicity~with~dotted } }
7296   }
7297   \bool_if:NTF \l_@@_tikz_rule_bool

```

```

7298 {
7299   \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
7300   \socket_assign_plug:nn { nicematrix / hsegment } { tikz }
7301 }
7302 {
7303   \bool_if:NTF \l_@@_dotted_rule_bool
7304   {
7305     \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
7306     \socket_assign_plug:nn { nicematrix / hsegment } { dotted }
7307   }
7308   {
7309     \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
7310     \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
7311   }
7312 }
7313 }

```

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.

```

7314 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
7315 {
7316   \int_compare:nNt \l_tmpa_tl > { #1 }
7317   {
7318     \int_compare:nNt \l_tmpa_tl < { #3 + 1 }
7319     {
7320       \int_compare:nNt \l_tmpb_tl > { #2 - 1 }
7321       {
7322         \int_compare:nNt \l_tmpb_tl < { #4 + 1 }
7323         { \bool_gset_false:N \g_tmpa_bool }
7324       }
7325     }
7326   }
7327 }

```

The same for vertical rules.

```

7328 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
7329 {
7330   \int_compare:nNt \l_tmpa_tl > { #1 - 1 }
7331   {
7332     \int_compare:nNt \l_tmpa_tl < { #3 + 1 }
7333     {
7334       \int_compare:nNt \l_tmpb_tl > { #2 }
7335       {
7336         \int_compare:nNt \l_tmpb_tl < { #4 + 1 }
7337         { \bool_gset_false:N \g_tmpa_bool }
7338       }
7339     }
7340   }
7341 }

7342 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
7343 {
7344   \int_compare:nNt \l_tmpb_tl > { #2 - 1 }
7345   {
7346     \int_compare:nNt \l_tmpb_tl < { #4 + 1 }
7347     {
7348       \int_compare:nNtTF \l_tmpa_tl = { #1 }
7349       { \bool_gset_false:N \g_tmpa_bool }
7350       {
7351         \int_compare:nNt \l_tmpa_tl = { #3 + 1 }
7352         { \bool_gset_false:N \g_tmpa_bool }

```

```

7353     }
7354   }
7355 }
7356 }
7357 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
7358 {
7359   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
7360   {
7361     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
7362     {
7363       \int_compare:nNnTF \l_tmpb_tl = { #2 }
7364       { \bool_gset_false:N \g_tmpa_bool }
7365       {
7366         \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
7367         { \bool_gset_false:N \g_tmpa_bool }
7368       }
7369     }
7370   }
7371 }

```

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.

```

7372 \cs_new_protected:Npn \@@_compute_corners:
7373 {
7374   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7375   { \@@_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`).

```

7376   \clist_clear:N \l_@@_corners_cells_clist
7377   \clist_map_inline:Nn \l_@@_corners_clist
7378   {
7379     \str_case:nnF { ##1 }
7380     {
7381       { NW }
7382       { \@@_compute_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
7383       { NE }
7384       { \@@_compute_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
7385       { SW }
7386       { \@@_compute_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
7387       { SE }
7388       { \@@_compute_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
7389       { N }
7390       { \@@_compute_corner_NS:nnnn \c@jCol 1 1 \c@iRow }
7391       { S }
7392       { \@@_compute_corner_NS:nnnn \c@jCol \c@iRow { -1 } 1 }
7393       { E }
7394       { \@@_compute_corner_EW:nnnn \c@iRow \c@jCol { -1 } 1 }
7395       { W }
7396       { \@@_compute_corner_EW:nnnn \c@iRow 1 1 \c@jCol }
7397     }
7398     { \@@_error:nn { bad~corner } { ##1 } }
7399   }

```


Even if the user has used the key `corners` the list of cells in the corners may be empty.

```
7400 \clist_if_empty:NF \l_@@_corners_cells_clist
7401 {
```

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.

```
7402 \tl_gput_right:Ne \g_@@_aux_tl
7403 {
7404 \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
7405 { \l_@@_corners_cells_clist }
7406 }
7407 }
7408 }
```

```
7409 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
7410 {
7411 \int_step_inline:nnn { #1 } { #3 }
7412 {
7413 \int_step_inline:nnn { #2 } { #4 }
7414 { \cs_set_nopar:cpn { @@ _ block _ ##1 - ###1 } { } }
7415 }
7416 }
```

```
7417 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
7418 {
7419 \cs_if_exist:cTF
7420 { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
7421 \prg_return_true:
7422 \prg_return_false:
7423 }
```

“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.

```
7424 \cs_new_protected:Npn \@@_compute_corner:nnnnnn #1 #2 #3 #4 #5 #6
7425 {
```

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.

```
7426 \bool_set_false:N \l_tmpa_bool
7427 \int_zero_new:N \l_@@_last_empty_row_int
7428 \int_set:Nn \l_@@_last_empty_row_int { #1 }
7429 \int_step_inline:nnnn { #1 } { #3 } { #5 }
7430 {
7431 \bool_lazy_or:nnTF
7432 {
7433 \cs_if_exist_p:c
7434 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
7435 }
```

```

7436     { \@@_if_in_block_p:nn { ##1 } { #2 } }
7437     { \bool_set_true:N \l_tmpa_bool }
7438     {
7439         \bool_if:NF \l_tmpa_bool
7440         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
7441     }
7442 }

```

Now, you determine the last empty cell in the row of number 1.

```

7443     \bool_set_false:N \l_tmpa_bool
7444     \int_zero_new:N \l_@@_last_empty_column_int
7445     \int_set:Nn \l_@@_last_empty_column_int { #2 }
7446     \int_step_inline:nnnn { #2 } { #4 } { #6 }
7447     {
7448         \bool_lazy_or:nnTF
7449         {
7450             \cs_if_exist_p:c
7451             { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
7452         }
7453         { \@@_if_in_block_p:nn { #1 } { ##1 } }
7454         { \bool_set_true:N \l_tmpa_bool }
7455     }
7456     \bool_if:NF \l_tmpa_bool
7457     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
7458 }
7459 }

```

Now, we loop over the rows.

```

7460     \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
7461     {

```

We treat the row number ##1 with another loop.

```

7462         \bool_set_false:N \l_tmpa_bool
7463         \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
7464         {
7465             \bool_lazy_or:nnTF
7466             { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7467             { \@@_if_in_block_p:nn { ##1 } { #####1 } }
7468             { \bool_set_true:N \l_tmpa_bool }
7469         }
7470         \bool_if:NF \l_tmpa_bool
7471         {
7472             \int_set:Nn \l_@@_last_empty_column_int { #####1 }
7473             \clist_put_right:Nn
7474             \l_@@_corners_cells_clist
7475             { ##1 - #####1 }
7476             \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7477         }
7478     }
7479 }
7480 }
7481 }

```

Of course, instead of the following lines, we could have use \prg_new_conditional:Npnn.

```

7482 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
7483 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }

```

Instead of the previous lines, we could have used \l_@@_corners_cells_clist but it's less efficient:
\clist_if_in:NeT \l_@@_corners_cells_clist { #1 } ...

```

7484 \cs_new_protected:Npn \@@_compute_corner_NS:nnnn #1 #2 #3 #4
7485 {
7486     \int_step_inline:nn { #1 }
7487     {

```

We will raise the flag: `\l_tmpa_bool` when we will find an non-empty cell.

```

7488     \bool_set_false:N \l_tmpa_bool
7489     \int_step_inline:nnnn { #2 } { #3 } { #4 }
7490     {
7491         \bool_lazy_or:nnTF
7492         { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 } }
7493         { \@@_if_in_block_p:nn { #####1 } { ##1 } }
7494         { \bool_set_true:N \l_tmpa_bool }
7495         {
7496             \bool_if:NF \l_tmpa_bool
7497             {
7498                 \clist_put_right:Nn
7499                 \l_@@_corners_cells_clist
7500                 { #####1 - ##1 }
7501                 \cs_set_nopar:cpn { @@ _ corner _ #####1 - ##1 } { }
7502             }
7503         }
7504     }
7505 }
7506 }

```

```

7507 \cs_new_protected:Npn \@@_compute_corner_EW:nnnn #1 #2 #3 #4
7508 {
7509     \int_step_inline:nn { #1 }
7510     {

```

We will raise the flag: `\l_tmpa_bool` when we will find an non-empty cell.

```

7511     \bool_set_false:N \l_tmpa_bool
7512     \int_step_inline:nnnn { #2 } { #3 } { #4 }
7513     {
7514         \bool_lazy_or:nnTF
7515         { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7516         { \@@_if_in_block_p:nn { ##1 } { #####1 } }
7517         { \bool_set_true:N \l_tmpa_bool }
7518         {
7519             \bool_if:NF \l_tmpa_bool
7520             {
7521                 \clist_put_right:Nn
7522                 \l_@@_corners_cells_clist
7523                 { ##1 - #####1 }
7524                 \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7525             }
7526         }
7527     }
7528 }
7529 }

```

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.

```

7530 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

7531 \keys_define:nn { nicematrix / NiceMatrixBlock }
7532 {

```

```

7533     auto-columns-width .code:n =
7534     {
7535         \bool_set_true:N \l_@@_block_auto_columns_width_bool
7536         \dim_gzero_new:N \g_@@_max_cell_width_dim
7537         \bool_set_true:N \l_@@_auto_columns_width_bool
7538     }
7539 }

7540 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
7541 {
7542     \int_gincr:N \g_@@_NiceMatrixBlock_int
7543     \dim_zero:N \l_@@_columns_width_dim
7544     \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
7545     \bool_if:NT \l_@@_block_auto_columns_width_bool
7546     {
7547         \cs_if_exist:cT
7548         { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
7549         {
7550             \dim_set:Nn \l_@@_columns_width_dim
7551             {
7552                 \use:c
7553                 { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7554             }
7555         }
7556     }
7557 }

```

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`).

```

7558 {
7559     \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.

```

7560     { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
7561     {
7562         \bool_if:NT \l_@@_block_auto_columns_width_bool
7563         {
7564             \iow_shipout:Nn \@mainaux \ExplSyntaxOn
7565             \iow_shipout:Ne \@mainaux
7566             {
7567                 \cs_gset:cpn
7568                 { @@_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.

```

7569             { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7570         }
7571         \iow_shipout:Nn \@mainaux \ExplSyntaxOff
7572     }
7573 }
7574 \ignorespacesafterend
7575 }

```

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).

```

7576 \cs_new_protected:Npn \@@_create_extra_nodes:
7577 {
7578   \bool_if:nTF \l_@@_medium_nodes_bool
7579   {
7580     \bool_if:NTF \l_@@_no_cell_nodes_bool
7581     { \@@_error:n { extra-nodes~with~no~cell~nodes } }
7582     {
7583       \bool_if:NTF \l_@@_large_nodes_bool
7584       \@@_create_medium_and_large_nodes:
7585       \@@_create_medium_nodes:
7586     }
7587   }
7588   {
7589     \bool_if:NT \l_@@_large_nodes_bool
7590     {
7591       \bool_if:NTF \l_@@_no_cell_nodes_bool
7592       { \@@_error:n { extra-nodes~with~no~cell~nodes } }
7593       \@@_create_large_nodes:
7594     }
7595   }
7596 }

```

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`.

```

7597 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7598 {
7599   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7600   {
7601     \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7602     \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7603     \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7604     \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7605   }
7606   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7607   {
7608     \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7609     \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7610     \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7611     \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7612   }

```

We begin the two nested loops over the rows and the columns of the array.

```

7613   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7614   {
7615     \int_step_variable:nnNn
7616     \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.

```

7617     {
7618         \cs_if_exist:cT
7619         { 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.

```

7620     {
7621         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7622         \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7623         { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7624         \seq_if_in:Nef \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7625         {
7626             \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7627             { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7628         }

```

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.

```

7629         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7630         \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7631         { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
7632         \seq_if_in:Nef \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7633         {
7634             \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7635             { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
7636         }
7637     }
7638 }
7639 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7640 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7641 {
7642     \dim_compare:nNnT
7643     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7644     {
7645         \@@_qpoint:n { row - \@@_i: - base }
7646         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7647         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7648     }
7649 }
7650 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7651 {
7652     \dim_compare:nNnT
7653     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7654     {
7655         \@@_qpoint:n { col - \@@_j: }
7656         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7657         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7658     }
7659 }
7660 }

```

Here is the command \@@_create_medium_nodes:. When this command is used, the “medium nodes” are created.

```

7661 \cs_new_protected:Npn \@@_create_medium_nodes:
7662 {
7663     \pgfpicture
7664     \pgfrememberpicturepositiononpagetrue
7665     \pgf@relevantforpicturesizefalse
7666     \@@_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”.

```

7667 \tl_set:Nn \l_@@_suffix_tl { -medium }
7668 \@@_create_nodes:
7669 \endpgfpicture
7670 }

```

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:`.

```

7671 \cs_new_protected:Npn \@@_create_large_nodes:
7672 {
7673   \pgfpicture
7674     \pgfrememberpicturepositiononpagetrue
7675     \pgf@relevantforpicturesizefalse
7676     \@@_computations_for_medium_nodes:
7677     \@@_computations_for_large_nodes:
7678     \tl_set:Nn \l_@@_suffix_tl { - large }
7679     \@@_create_nodes:
7680   \endpgfpicture
7681 }

7682 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7683 {
7684   \pgfpicture
7685     \pgfrememberpicturepositiononpagetrue
7686     \pgf@relevantforpicturesizefalse
7687     \@@_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”.

```

7688 \tl_set:Nn \l_@@_suffix_tl { - medium }
7689 \@@_create_nodes:
7690 \@@_computations_for_large_nodes:
7691 \tl_set:Nn \l_@@_suffix_tl { - large }
7692 \@@_create_nodes:
7693 \endpgfpicture
7694 }

```

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.

```

7695 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7696 {
7697   \int_set:Nn \l_@@_first_row_int 1
7698   \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`.

```

7699 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7700 {
7701   \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7702   {
7703     (
7704       \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7705       \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7706     )
7707     / 2
7708   }

```

¹⁵If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7709     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7710     { l_@@_row _ \@@_i: _min_dim }
7711 }
7712 \int_step_variable:nnN { \c@jCol - 1 } \@@_j:
7713 {
7714     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7715     {
7716         (
7717             \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7718             \dim_use:c
7719             { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7720         )
7721         / 2
7722     }
7723     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7724     { l_@@_column _ \@@_j: _ max _ dim }
7725 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7726     \dim_sub:cn
7727     { l_@@_column _ 1 _ min _ dim }
7728     \l_@@_left_margin_dim
7729     \dim_add:cn
7730     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7731     \l_@@_right_margin_dim
7732 }

```

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).

```

7733 \cs_new_protected:Npn \@@_create_nodes:
7734 {
7735     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7736     {
7737         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7738         {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7739     \@@_pgf_rect_node:nnnnn
7740     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7741     { \dim_use:c { l_@@_column _ \@@_j: _min_dim } }
7742     { \dim_use:c { l_@@_row _ \@@_i: _min_dim } }
7743     { \dim_use:c { l_@@_column _ \@@_j: _max_dim } }
7744     { \dim_use:c { l_@@_row _ \@@_i: _max_dim } }
7745     \str_if_empty:NF \l_@@_name_str
7746     {
7747         \pgfnodealias
7748         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7749         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7750     }
7751 }
7752 }
7753 \int_step_inline:nn \c@iRow
7754 {
7755     \pgfnodealias
7756     { \@@_env: - ##1 - last \l_@@_suffix_tl }
7757     { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7758 }
7759 \int_step_inline:nn \c@jCol
7760 {

```



```

7761 \pgfnodealias
7762 { \@@_env: - last - ##1 \l_@@_suffix_tl }
7763 { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7764 }
7765 \pgfnodealias
7766 { \@@_env: - last - last \l_@@_suffix_tl }
7767 { \@@_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 .

```

7768 \seq_map_pairwise_function:NNN
7769 \g_@@_multicolumn_cells_seq
7770 \g_@@_multicolumn_sizes_seq
7771 \@@_node_for_multicolumn:nn
7772 }

7773 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7774 {
7775   \cs_set_nopar:Npn \@@_i: { #1 }
7776   \cs_set_nopar:Npn \@@_j: { #2 }
7777 }

```

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”).

```

7778 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7779 {
7780   \@@_extract_coords_values: #1 \q_stop
7781   \@@_pgf_rect_node:nnnnn
7782   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7783   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7784   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7785   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: + #2 - 1 } _ max _ dim } }
7786   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7787   \str_if_empty:NF \l_@@_name_str
7788   {
7789     \pgfnodealias
7790     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7791     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7792   }
7793 }

```

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).

```

7794 \keys_define:nn { nicematrix / BlockFirstPass }
7795 {
7796   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7797   \bool_set_true:N \l_@@_p_block_bool ,
7798   j .value_forbidden:n = true ,
7799   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7800   l .value_forbidden:n = true ,

```

```

7801 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7802 r .value_forbidden:n = true ,
7803 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7804 c .value_forbidden:n = true ,
7805 L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7806 L .value_forbidden:n = true ,
7807 R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7808 R .value_forbidden:n = true ,
7809 C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7810 C .value_forbidden:n = true ,
7811 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7812 t .value_forbidden:n = true ,
7813 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7814 T .value_forbidden:n = true ,
7815 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7816 b .value_forbidden:n = true ,
7817 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7818 B .value_forbidden:n = true ,
7819 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7820 m .value_forbidden:n = true ,
7821 v-center .meta:n = m ,
7822 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7823 p .value_forbidden:n = true ,
7824 respect-arraystretch .code:n =
7825     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7826 respect-arraystretch .value_forbidden:n = true ,

```

The following key is no longer documented in `nicematrix.tex` and `nicematrix-french.tex`. It should be considered as deprecated.

```

7827 color .code:n =
7828     \@@_color:n { #1 }
7829     \tl_set_rescan:Nnn
7830         \l_@@_draw_tl
7831         { \char_set_catcode_other:N ! }
7832         { #1 } ,
7833 color .value_required:n = true ,
7834 }

```

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.

```

7835 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7836 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7837 {

```

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).

```

7838 \tl_if_blank:nTF { #2 }
7839 { \@@_Block_ii:nnnnn 1 1 }
7840 {
7841     \tl_if_in:nnTF { #2 } { - }
7842     {
7843         \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7844         \@@_Block_i_czech:w \@@_Block_i:w
7845         #2 \q_stop
7846     }
7847     {
7848         \@@_error:nn { Bad~argument~for~Block } { #2 }
7849         \@@_Block_ii:nnnnn 1 1
7850     }
7851 }
7852 { #1 } { #3 } { #4 }

```

```

7853 \ignorespaces
7854 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7855 \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`.

```

7856 {
7857   \char_set_catcode_active:N -
7858   \cs_new:Npn \@@_Block_i-czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7859 }

```

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.

```

7860 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7861 {

```

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).

```

7862   \bool_lazy_or:nnTF
7863     { \tl_if_blank_p:n { #1 } }
7864     { \str_if_eq_p:ee { * } { #1 } }
7865     { \int_set:Nn \l_tmpa_int { 100 } }
7866     { \int_set:Nn \l_tmpa_int { #1 } }
7867   \bool_lazy_or:nnTF
7868     { \tl_if_blank_p:n { #2 } }
7869     { \str_if_eq_p:ee { * } { #2 } }
7870     { \int_set:Nn \l_tmpb_int { 100 } }
7871     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7872   \int_compare:nNnTF \l_tmpb_int = 1
7873   {
7874     \tl_if_empty:NNTF \l_@@_hpos_cell_tl
7875       { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7876       { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7877   }
7878   { \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.

```

7879   \keys_set_known:nn { nicematrix / BlockFirstPass } { #3 }
7880   \tl_set:Ne \l_tmpa_tl
7881   {
7882     { \int_use:N \c@iRow }
7883     { \int_use:N \c@jCol }
7884     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7885     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7886   }

```

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).

```

7887 \bool_set_false:N \l_tmpa_bool
7888 \bool_if:NT \l_@@_amp_in_blocks_bool
\tl_if_in:nnT is slightly faster than \str_if_in:nnT.
7889 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7890 \bool_case:nF
7891 {
7892 \l_tmpa_bool { \@@_Block_vii:eennn }
7893 \l_@@_p_block_bool { \@@_Block_vi:eennn }

```

For the blocks mono-column, we will compose right away in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a `X` column, we should not do that since the width is determined by another way. This should be the same for the `p`, `m` and `b` columns and we should modify that point. However, for the `X` column, it's imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

7894 \l_@@_X_bool { \@@_Block_v:eennn }
7895 { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eennn }
7896 { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7897 { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7898 }
7899 { \@@_Block_v:eennn }
7900 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7901 }

```

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.

```

7902 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7903 {
7904 \int_gincr:N \g_@@_block_box_int
7905 \cs_set_eq:NN \cellcolor \@@_cellcolor_error
7906 \cs_set_eq:NN \rowcolor \@@_rowcolor_error
7907 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7908 {
7909 \tl_gput_right:Ne \g_@@_rules_tl
7910 {
7911 \@@_draw_diagbox:nnnnnn
7912 { \int_use:N \c@iRow }
7913 { \int_use:N \c@jCol }
7914 { \int_eval:n { \c@iRow + #1 - 1 } }
7915 { \int_eval:n { \c@jCol + #2 - 1 } }
7916 { \g_@@_row_style_tl \exp_not:n { ##1 } }
7917 { \g_@@_row_style_tl \exp_not:n { ##2 } }
7918 }
7919 }
7920 \box_gclear_new:c
7921 { 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!

```

7922 \hbox_gset:cn

```

```

7923 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7924 {

```

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`).

```

7925 \tl_if_empty:NTF \l_@@_color_tl
7926 { \int_compare:nNnT { #2 } = 1 \set@color }
7927 { \@@_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`.

```

7928 \int_compare:nNnT { #1 } = 1
7929 {
7930 \int_if_zero:nTF \c@iRow
7931 {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7932 \cs_set_eq:NN \Block \@@_NullBlock:
7933 \l_@@_code_for_first_row_tl
7934 }
7935 {
7936 \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7937 {
7938 \cs_set_eq:NN \Block \@@_NullBlock:
7939 \l_@@_code_for_last_row_tl
7940 }
7941 }
7942 \g_@@_row_style_tl
7943 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7944 \@@_reset_arraystretch:
7945 \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7946 #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.).

```

7947 \@@_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}`.

```

7948     \bool_if:NTF \l_@@_tabular_bool
7949     {
7950         \bool_lazy_all:nTF
7951         {
7952             { \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.

```

7953         { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7954         { ! \g_@@_rotate_bool }
7955     }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7956     {
7957         \use:e
7958         {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7959             \exp_not:N \begin { minipage }
7960             [ \str_lowercase:f \l_@@_vpos_block_str ]
7961             { \l_@@_col_width_dim }
7962             \str_case:on \l_@@_hpos_block_str
7963             { c \centering r \raggedleft l \raggedright }
7964         }
7965         #5
7966     \end { minipage }
7967 }

```

In the other cases, we use a `{tabular}`.

```

7968     {
7969         \use:e
7970         {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7971             \exp_not:N \begin { tabular }
7972             [ \str_lowercase:f \l_@@_vpos_block_str ]
7973             { @ { } \l_@@_hpos_block_str @ { } }
7974         }
7975         #5
7976     \end { tabular }
7977 }
7978 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7979     {
7980         $ % $
7981         \bool_if:NT \l_@@_small_bool
7982         {
7983             \def \arraystretch { 0.47 }
7984             \dim_set:Nn \arraycolsep { 1.45 pt }
7985         }
7986         \use:e
7987         {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7988             \exp_not:N \begin { array }
7989             [ \str_lowercase:f \l_@@_vpos_block_str ]
7990             { @ { } \l_@@_hpos_block_str @ { } }
7991         }

```

```

7992         \@@_tuning_key_small:
7993         #5
7994     \end { array }
7995     $ % $
7996 }
7997 }

```

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).

```

7998     \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.

```

7999     \int_compare:nNnT { #2 } = 1
8000     {
8001         \dim_gset:Nn \g_@@_blocks_wd_dim
8002         {
8003             \dim_max:nn
8004             \g_@@_blocks_wd_dim
8005             {
8006                 \box_wd:c
8007                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8008             }
8009         }
8010     }

```

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.

```

8011     \int_compare:nNnT { #1 } = 1
8012     {
8013         \bool_lazy_any:nT
8014         {
8015             { \str_if_empty_p:N \l_@@_vpos_block_str }
8016             { \str_if_eq_p:ee t \l_@@_vpos_block_str }
8017             { \str_if_eq_p:ee b \l_@@_vpos_block_str }
8018         }
8019         \@@_adjust_blocks_ht_dp:
8020     }
8021     \seq_gput_right:Ne \g_@@_blocks_seq
8022     {
8023         \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.

```

8024     {
8025         \exp_not:n { #3 } ,
8026         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

8027     \bool_if:NT \g_@@_rotate_bool
8028     {
8029         \bool_if:NTF \g_@@_rotate_c_bool
8030         { m }
8031         {
8032             \int_compare:nNnT \c@iRow = \l_@@_last_row_int
8033             { T }
8034         }
8035     }
8036 }

```

```

8037     {
8038         \box_use_drop:c
8039         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8040     }
8041 }
8042 \bool_set_false:N \g_@@_rotate_c_bool
8043 }
8044 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
8045 {
8046     \dim_gset:Nn \g_@@_blocks_ht_dim
8047     {
8048         \dim_max:nn
8049         \g_@@_blocks_ht_dim
8050         {
8051             \box_ht:c
8052             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8053         }
8054     }
8055     \dim_gset:Nn \g_@@_blocks_dp_dim
8056     {
8057         \dim_max:nn
8058         \g_@@_blocks_dp_dim
8059         {
8060             \box_dp:c
8061             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8062         }
8063     }
8064 }
8065 \cs_new:Npn \@@_adjust_hpos_rotate:
8066 {
8067     \bool_if:NT \g_@@_rotate_bool
8068     {
8069         \str_set:Ne \l_@@_hpos_block_str
8070         {
8071             \bool_if:NTF \g_@@_rotate_c_bool
8072             c
8073             {
8074                 \str_case:onF \l_@@_vpos_block_str
8075                 { b l B l t r T r }
8076                 {
8077                     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
8078                     r
8079                     l
8080                 }
8081             }
8082         }
8083     }
8084 }
8085 \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*.

```

8086 \cs_new_protected:Npn \@@_rotate_box_of_block:
8087 {
8088     \box_grotate:cn
8089     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8090     { \bool_if:NTF \g_@@_rotate_minus_bool { -90 } { 90 } }
8091     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
8092     {
8093         \vbox_gset_top:cn

```



```

8094     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8095     {
8096       \skip_vertical:n { 0.8 ex }
8097       \box_use:c
8098       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8099     }
8100   }
8101 \bool_if:NT \g_@@_rotate_c_bool
8102 {
8103   \hbox_gset:cn
8104   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8105   {
8106     \IfFormatAtLeastTF { 2026-04-01 }
8107     {
8108       \vbox_center:n
8109       {
8110         \box_use:c
8111         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8112       }
8113     }
8114     {
8115       $ % $
8116       \vcenter
8117       {
8118         \box_use:c
8119         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8120       }
8121       $ % $
8122     }
8123   }
8124 }
8125 }

```

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.

```

8126 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
8127 {
8128   \seq_gput_right:Ne \g_@@_blocks_seq
8129   {
8130     \l_tmpa_tl
8131     { \exp_not:n { #3 } }
8132     {
8133       \bool_if:NTF \l_@@_tabular_bool
8134       {
8135         {

```

The following command will be no-op when `respect-arraystretch` is in force.

```

8136 \@@_reset_arraystretch:
8137 \exp_not:n
8138 {
8139   \dim_zero:N \extrarowheight
8140   #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).

```

8141         \tag_if_active:T { \tag_stop:n { table } }
8142         \use:e
8143         {
8144             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
8145             { @ { } \l_@@_hpos_block_str @ { } }
8146         }
8147         #5
8148         \end { tabular }
8149     }
8150 }
8151 }

```

When we are *not* in an environment {NiceTabular} (or similar).

```

8152     {
8153     {

```

The following will be no-op when respect-arraystretch is in force.

```

8154         \@@_reset_arraystretch:
8155         \exp_not:n
8156         {
8157             \dim_zero:N \extrarowheight
8158             #4
8159             $ % $
8160             \use:e
8161             {
8162                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
8163                 { @ { } \l_@@_hpos_block_str @ { } }
8164             }
8165             \@@_tuning_key_small:
8166             #5
8167             \end { array }
8168             $ % $
8169         }
8170     }
8171 }
8172 }
8173 }
8174 }
8175 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a \Block which uses the key p.

```

8176 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
8177 {
8178     \seq_gput_right:Ne \g_@@_blocks_seq
8179     {
8180         \l_tmpa_tl
8181         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

8182         { { \exp_not:n { #4 #5 } } }
8183     }
8184 }
8185 \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.

```

8186 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
8187 {
8188     \seq_gput_right:Ne \g_@@_blocks_seq
8189     {
8190         \l_tmpa_tl
8191         { \exp_not:n { #3 } }
8192         { \exp_not:n { #4 #5 } }
8193     }

```

```

8194 }
8195 \cs_generate_variant:Nn \l_@@_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).

```

8196 \keys_define:nn { nicematrix / Block }
8197 {
8198   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
8199   &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

8200   tikz .code:n =
8201     \IfPackageLoadedTF { tikz }
8202     { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
8203     { \@@_error:n { tikz~key~without~tikz } } ,
8204   tikz .value_required:n = true ,
8205   fill .code:n =
8206     \tl_set_rescan:Nnn
8207     \l_@@_fill_tl
8208     { \char_set_catcode_other:N ! }
8209     { #1 } ,
8210   fill .value_required:n = true ,

```

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

```

8211   opacity .tl_set:N = \l_@@_opacity_tl ,
8212   opacity .value_required:n = true ,
8213   draw .code:n =
8214     \tl_set_rescan:Nnn
8215     \l_@@_draw_tl
8216     { \char_set_catcode_other:N ! }
8217     { #1 } ,
8218   draw .default:n = default ,
8219   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8220   rounded-corners .default:n = 4 pt ,
8221   color .code:n =
8222     \@@_color:n { #1 }
8223     \tl_set_rescan:Nnn
8224     \l_@@_draw_tl
8225     { \char_set_catcode_other:N ! }
8226     { #1 } ,
8227   borders .clist_set:N = \l_@@_borders_clist ,
8228   borders .default:n = all ,
8229   hvlines .meta:n = { vlines , hlines } ,
8230   vlines .bool_set:N = \l_@@_vlines_block_bool ,
8231   hlines .bool_set:N = \l_@@_hlines_block_bool ,
8232   rules/width .dim_set:N = \arrayrulewidth ,
8233   rules/color .tl_set:N = \l_@@_rules_color_tl ,
8234   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,

```

The key `line-width` is now deprecated (replaced by `rules/width`).

```

8235   line-width .dim_set:N = \arrayrulewidth , % deprecated

```

Some keys have not a property `.value_required:n` (or similar) because they are in `BlockFirstPass`.

```

8236   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
8237             \bool_set_true:N \l_@@_p_block_bool ,
8238   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
8239   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
8240   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
8241   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
8242             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8243   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
8244             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,

```

```

8245 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
8246         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8247 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
8248 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
8249 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
8250 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
8251 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
8252 m .value_forbidden:n = true ,
8253 v-center .meta:n = m ,
8254 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
8255 p .value_forbidden:n = true ,
8256 name .str_set:N = \l_@@_block_name_str ,
8257 name .value_required:n = true ,
8258 respect-arraystretch .code:n =
8259     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
8260 respect-arraystretch .value_forbidden:n = true ,
8261 transparent .bool_set:N = \l_@@_transparent_bool ,
8262 unknown .code:n =
8263     \@@_unknown_key:nn
8264         { nicematrix / Block }
8265         { Unknown-key-for-Block }
8266 }

```

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.

```

8267 \cs_new_protected:Npn \@@_draw_blocks:
8268 {
8269     \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
8270     \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
8271 }
8272 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
8273 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

8274 \int_zero:N \l_@@_last_row_int
8275 \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”).

```

8276 \int_compare:nNnTF { #3 } > { 98 }
8277 { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
8278 { \int_set:Nn \l_@@_last_row_int { #3 } }
8279 \int_compare:nNnTF { #4 } > { 98 }
8280 { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
8281 { \int_set:Nn \l_@@_last_col_int { #4 } }
8282 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
8283 {
8284     \bool_lazy_and:nnTF
8285         \l_@@_preamble_bool
8286         {
8287             \int_compare_p:n
8288                 { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
8289         }
8290     {
8291         \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
8292         \@@_msg_redirect_name:nn { Block-too-large-2 } { none }

```

```

8293         \@@_msg_redirect_name:nn { columns-not-used } { none }
8294     }
8295     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8296 }
8297 {
8298     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
8299     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8300     {

```

We expand the four first arguments of `\@@_Block_v:nnnnnn`.

```

8301         \use:e
8302         {
8303             \@@_Block_v:nnnnnn
8304             { #1 }
8305             { #2 }
8306             { \int_use:N \l_@@_last_row_int }
8307             { \int_use:N \l_@@_last_col_int }
8308         }
8309         { #5 }
8310         { #6 }
8311     }
8312 }
8313 }

```

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.

```

8314 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
8315 {

```

The group is for the keys.

```

8316     \group_begin:
8317     \int_compare:nNnT { #1 } = { #3 }
8318     { \str_set:Nn \l_@@_vpos_block_str { t } }
8319     \keys_set:nn { nicematrix / Block } { #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`.

```

8320     \bool_if:NT \l_@@_amp_in_blocks_bool
8321     { \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool } }

8322     \bool_lazy_and:nnT
8323     \l_@@_vlines_block_bool
8324     { ! \l_@@_ampersand_bool }
8325     {
8326         \tl_gput_right:Ne \g_@@_rules_tl
8327         {
8328             \@@_vlines_block:nnnnnn
8329             { \dim_use:N \arrayrulewidth }
8330             { \l_@@_rules_color_tl }
8331             { #1 } { #2 } { #3 } { #4 }
8332         }
8333     }
8334     \bool_if:NT \l_@@_hlines_block_bool
8335     {
8336         \tl_gput_right:Ne \g_@@_rules_tl
8337         {
8338             \@@_hlines_block:nnnnnn
8339             { \dim_use:N \arrayrulewidth }
8340             { \l_@@_rules_color_tl }
8341             { #1 } { #2 } { #3 } { #4 }
8342         }
8343     }

```

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).

```

8344 \bool_if:NF \l_@@_transparent_bool
8345 {
8346   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8347   { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
8348 }

```

```

8349 \tl_if_empty:NF \l_@@_draw_tl
8350 {
8351   \tl_gput_right:Nn \g_@@_rules_tl
8352   {
8353     \@@_stroke_block:nnnnn

```

#5 are the options

```

8354       { #5 } { #1 } { #2 } { #3 } { #4 }
8355     }
8356     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
8357     { { #1 } { #2 } { #3 } { #4 } }
8358   }
8359 \clist_if_empty:NF \l_@@_borders_clist
8360 {
8361   \tl_gput_right:Nn \g_nicematrix_code_after_tl
8362   {
8363     \@@_stroke_borders_block:nnnnn
8364     { #5 } { #1 } { #2 } { #3 } { #4 }
8365   }
8366 }
8367 \tl_if_empty:NF \l_@@_fill_tl
8368 {
8369   \@@_add_opacity_to_fill:
8370   \tl_gput_right:Ne \g_@@_pre_code_before_tl
8371   {
8372     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
8373     { #1 - #2 }
8374     { #3 - #4 }
8375     { \dim_use:N \l_@@_rounded_corners_dim }
8376   }
8377 }
8378 \seq_if_empty:NF \l_@@_tikz_seq
8379 {
8380   \tl_gput_right:Ne \g_nicematrix_code_before_tl
8381   {
8382     \@@_block_tikz:nnnnn
8383     { \seq_use:Nn \l_@@_tikz_seq { , } }
8384     { #1 } { #2 } { #3 } { #4 }

```

We will have in that last field a list of lists of TikZ keys.

```

8385   }
8386 }
8387 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
8388 {
8389   \tl_gput_right:Ne \g_@@_rules_tl
8390   {
8391     \@@_draw_diagbox:nnnnnn
8392     { #1 } { #2 } { #3 } { #4 }
8393     { \exp_not:n { ##1 } }
8394     { \exp_not:n { ##2 } }
8395   }
8396 }

```

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
three	four	two
six	seven	five
		eight

We highlight the node `1-1-block-short`

our block	one
three	two
four	five
six	eight

The construction of the node corresponding to the merged cells.

```
8397 \pgfpicture
8398 \pgfrememberpicturepositiononpagetrue
8399 \pgf@relevantforpicturesizefalse
8400 \@@_qpoint:n { row - #1 }
8401 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8402 \@@_qpoint:n { col - #2 }
8403 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8404 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
8405 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8406 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8407 \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.

```
8408 \@@_pgf_rect_node:nnnnn
8409 { \@@_env: - #1 - #2 - block }
8410 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8411 \str_if_empty:NF \l_@@_block_name_str
8412 {
8413   \pgfnodealias
8414   { \@@_env: - \l_@@_block_name_str }
8415   { \@@_env: - #1 - #2 - block }
8416   \str_if_empty:NF \l_@@_name_str
8417   {
8418     \pgfnodealias
8419     { \l_@@_name_str - \l_@@_block_name_str }
8420     { \@@_env: - #1 - #2 - block }
8421   }
8422 }
```

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.

```
8423 \bool_if:NF \l_@@_hpos_of_block_cap_bool
8424 {
8425   \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.

```
8426 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8427 {
```

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.

```

8428     \cs_if_exist:cT
8429     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8430     {
8431         \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8432         {
8433             \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
8434             \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
8435         }
8436     }
8437 }

```

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.

```

8438     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
8439     {
8440         \@@_qpoint:n { col - #2 }
8441         \dim_set_eq:NN \l_tmpb_dim \pgf@x
8442     }
8443     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
8444     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8445     {
8446         \cs_if_exist:cT
8447         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8448         {
8449             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8450             {
8451                 \pgfpointanchor
8452                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8453                 { east }
8454                 \dim_set:Nn \l_@@_tmpd_dim
8455                 { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
8456             }
8457         }
8458     }
8459     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
8460     {
8461         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8462         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8463     }
8464     \@@_pgf_rect_node:nnnnn
8465     { \@@_env: - #1 - #2 - block - short }
8466     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8467 }

```

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.

```

8468     \bool_if:NT \l_@@_medium_nodes_bool
8469     {
8470         \@@_pgf_rect_node:nnn
8471         { \@@_env: - #1 - #2 - block - medium }
8472         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
8473         {
8474             \pgfpointanchor
8475             { \@@_env:
8476               - \int_use:N \l_@@_last_row_int
8477               - \int_use:N \l_@@_last_col_int - medium
8478             }
8479             { south-east }
8480         }
8481     }
8482     \endpgfpicture
8483

```


`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand `&`.

```

8484 \bool_if:NTF \l_@@_ampersand_bool
8485 {
8486   \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
8487   \int_zero_new:N \l_@@_split_int
8488   \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).

```

8489   \int_set:Nn \l_@@_first_row_int { #1 }
8490   \int_set:Nn \l_@@_first_col_int { #2 }
8491   \int_set:Nn \l_@@_last_row_int { #3 }
8492   \int_set:Nn \l_@@_last_col_int { #4 }

8493   \pgfpicture
8494   \pgfrememberpicturepositiononpagetrue
8495   \pgf@relevantforpicturesizefalse
8496   \@@_qpoint:n { row - #1 }
8497   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8498   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8499   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
8500   \@@_qpoint:n { col - #2 }
8501   \dim_set_eq:NN \l_tmpa_dim \pgf@x
8502   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8503   \dim_set:Nn \l_tmpb_dim
8504     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
8505   \bool_lazy_or:nnT
8506     \l_@@_vlines_block_bool
8507     { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
8508     {
8509       \int_step_inline:nn { \l_@@_split_int - 1 }
8510       {
8511         \pgfpathmoveto
8512         {
8513           \pgfpoint
8514             { \l_tmpa_dim + ##1 \l_tmpb_dim }
8515             \l_@@_tmpc_dim
8516         }
8517         \pgfpathlineto
8518         {
8519           \pgfpoint
8520             { \l_tmpa_dim + ##1 \l_tmpb_dim }
8521             \l_@@_tmpd_dim
8522         }
8523         \CT@arc@
8524         \pgfsetlinewidth { 1.1 \arrayrulewidth }
8525         \pgfsetrectcap
8526         \pgfusepathqstroke
8527       }
8528     }
8529   \cs_set_eq:NN \cellcolor \@@_subcellcolor
8530   \int_zero_new:N \l_@@_split_i_int

8531   \str_if_eq:eeTF \l_@@_vpos_block_str T
8532   {
8533     \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
8534     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8535   }
8536   {
8537     \str_if_eq:eeTF \l_@@_vpos_block_str B
8538     {
8539       \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
8540       \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8541     }

```

```

8542     {
8543         \bool_lazy_or:nnTF
8544         { \int_compare_p:nNn { #1 } = { #3 } }
8545         { \str_if_eq_p:ee \l_@@_vpos_block_str t }
8546     }
8547     \@@_qpoint:n { row - #1 - base }
8548     \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8549 }
8550 {
8551     \str_if_eq:eeTF \l_@@_vpos_block_str b
8552     {
8553         \@@_qpoint:n { row - #3 - base }
8554         \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8555     }
8556     {
8557         \@@_qpoint:n { #1 - #2 - block }
8558         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8559     }
8560 }
8561 }
8562 }
8563 \int_step_inline:nn \l_@@_split_int
8564 {
8565     \group_begin:

```

The counter `\l_@@_split_i_int` is only for the command `\@@_subcellcolor`.

```

8566     \int_set:Nn \l_@@_split_i_int { ##1 }
8567     \dim_set:Nn \col@sep
8568     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
8569     \pgftransformshift
8570     {
8571         \pgfpoint
8572         {
8573             \l_tmpa_dim + ##1 \l_tmpb_dim -
8574             \str_case:on \l_@@_hpos_block_str
8575             {
8576                 l { \l_tmpb_dim + \col@sep }
8577                 c { 0.5 \l_tmpb_dim }
8578                 r { \col@sep }
8579             }
8580         }
8581         { \l_@@_tmpc_dim }
8582     }
8583     \pgfset { inner~sep = \c_zero_dim }
8584     \pgfnode
8585     { rectangle }
8586     {
8587         \str_if_eq:eeTF T \l_@@_vpos_block_str
8588         {
8589             \str_case:on \l_@@_hpos_block_str
8590             {
8591                 l { north-west }
8592                 c { north }
8593                 r { north-east }
8594             }
8595         }
8596         {
8597             \str_if_eq:eeTF B \l_@@_vpos_block_str
8598             {
8599                 \str_case:on \l_@@_hpos_block_str
8600                 {
8601                     l { south-west }
8602                     c { south }
8603                     r { south-east }

```

```

8604     }
8605   }
8606   {
8607     \bool_lazy_any:nTF
8608     {
8609       { \int_compare_p:nNn { #1 } = { #3 } }
8610       { \str_if_eq_p:ee t \l_@@_vpos_block_str }
8611       { \str_if_eq_p:ee b \l_@@_vpos_block_str }
8612     }
8613     {
8614       \str_case:on \l_@@_hpos_block_str
8615       {
8616         l { base~west }
8617         c { base }
8618         r { base~east }
8619       }
8620     }
8621     {
8622       \str_case:on \l_@@_hpos_block_str
8623       {
8624         l { west }
8625         c { center }
8626         r { east }
8627       }
8628     }
8629   }
8630 }
8631 }
8632 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8633 \group_end:
8634 }
8635 \endpgfpicture
8636 }

```

Now the case where there is no ampersand & in the content of the block.

```

8637 {
8638   \bool_if:NTF \l_@@_p_block_bool
8639   {

```

When the final user has used the key p, we have to compute the width.

```

8640   \pgfpicture
8641   \pgfrememberpicturepositiononpagetrue
8642   \pgf@relevantforpicturesizefalse
8643   \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8644   {
8645     \@@_qpoint:n { col - #2 }
8646     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8647     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8648   }
8649   {
8650     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8651     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8652     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8653   }
8654   \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8655 \endpgfpicture
8656 \hbox_set:Nn \l_@@_cell_box
8657 {
8658   \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8659     { \g_tmpb_dim }
8660     \str_case:on \l_@@_hpos_block_str
8661     { c \centering r \raggedleft l \raggedright j { } }
8662     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8663     #6

```

```

8664         \end { minipage }
8665     }
8666 }
8667 {
8668     \hbox_set:Nn \l_@@_cell_box
8669     {
8670         \set@color
8671         \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8672         #6
8673     }
8674 }
8675 \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.

```

8676 \pgfpicture
8677 \pgfrememberpicturepositiononpagetrue
8678 \pgf@relevantforpicturesizefalse
8679 \bool_lazy_any:nTF
8680 {
8681     { \str_if_empty_p:N \l_@@_vpos_block_str }
8682     { \str_if_eq_p:ee c \l_@@_vpos_block_str }
8683     { \str_if_eq_p:ee T \l_@@_vpos_block_str }
8684     { \str_if_eq_p:ee B \l_@@_vpos_block_str }
8685 }
8686 {

```

If we are in the “first column”, we must put the block as if it was with the key `r`.

```

8687     \int_if_zero:nT { #2 } { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_r_str }

```

If we are in the “last column”, we must put the block as if it was with the key `l`.

```

8688     \bool_if:nT \g_@@_last_col_found_bool
8689     {
8690         \int_compare:nNnT { #2 } = \g_@@_col_total_int
8691         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8692     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8693     \tl_set:Ne \l_tmpa_tl
8694     {
8695         \str_case:on \l_@@_vpos_block_str
8696         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8697         { } {
8698             \str_case:on \l_@@_hpos_block_str
8699             {
8700                 c { center }
8701                 l { west }
8702                 r { east }
8703                 j { center }
8704             }
8705         }
8706         c {
8707             \str_case:on \l_@@_hpos_block_str
8708             {
8709                 c { center }
8710                 l { west }
8711                 r { east }
8712                 j { center }
8713             }
8714         }

```

```

8715     }
8716     T {
8717         \str_case:on \l_@@_hpos_block_str
8718         {
8719             c { north }
8720             l { north-west }
8721             r { north-east }
8722             j { north }
8723         }
8724
8725     }
8726     B {
8727         \str_case:on \l_@@_hpos_block_str
8728         {
8729             c { south }
8730             l { south-west }
8731             r { south-east }
8732             j { south }
8733         }
8734
8735     }
8736 }
8737
8738 \pgftransformshift
8739 {
8740     \pgfpointanchor
8741     {
8742         \@@_env: - #1 - #2 - block
8743         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8744     }
8745     \l_tmpa_tl
8746 }
8747 \pgfset { inner~sep = \c_zero_dim }
8748 \pgfnode
8749 { rectangle }
8750 \l_tmpa_tl
8751 { \box_use_drop:N \l_@@_cell_box } { } { }
8752 }

```

End of the case when `\l_@@_vpos_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```

8753 {
8754     \pgfextracty \l_tmpa_dim
8755     {
8756         \@@_qpoint:n
8757         {
8758             row - \str_if_eq:eeTF b \l_@@_vpos_block_str { #3 } { #1 }
8759             - base
8760         }
8761     }
8762     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

8763 \pgfpointanchor
8764 {
8765     \@@_env: - #1 - #2 - block
8766     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8767 }
8768 {
8769     \str_case:on \l_@@_hpos_block_str
8770     {
8771         c { center }
8772         l { west }
8773         r { east }

```

```

8774         j { center }
8775     }
8776 }

We put the label of the block which has been composed in \l_@@_cell_box.

8777     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8778     \pgfset { inner~sep = \c_zero_dim }
8779     \pgfnode
8780     { rectangle }
8781     {
8782         \str_case:on \l_@@_hpos_block_str
8783         {
8784             c { base }
8785             l { base~west }
8786             r { base~east }
8787             j { base }
8788         }
8789     }
8790     { \box_use_drop:N \l_@@_cell_box } { } { }
8791 }

8792 \endpgfpicture
8793 }
8794 \group_end:
8795 }
8796 \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).

```

8797 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8798 {
8799     \tl_if_empty:NF \l_@@_opacity_tl
8800     {
8801         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8802             {
8803                 \tl_set:Nc \l_@@_fill_tl
8804                 {
8805                     [ opacity = \l_@@_opacity_tl ,
8806                     \tl_tail:o \l_@@_fill_tl
8807                 }
8808             }
8809             {
8810                 \tl_set:Nc \l_@@_fill_tl
8811                 { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8812             }
8813         }
8814     }

```

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*.

```

8815 \cs_new_protected:Npn \@@_stroke_block:nnnnn #1 #2 #3 #4 #5
8816 {
8817     \group_begin:
8818     \tl_clear:N \l_@@_draw_tl
8819     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8820     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8821     \tl_if_empty:NF \l_@@_draw_tl
8822     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8823     \tl_if_eq:NnTF \l_@@_draw_tl { default }

```

```

8824         \CT@arc@
8825         { \@@_color:o \l_@@_draw_tl }
8826     }

```

The following code can't be put just before the `\pgfusepath { stroke }` (it's too late).

```

8827 \pgfsetcornersarced
8828 { \pgfpoint \l_@@_rounded_corners_dim \l_@@_rounded_corners_dim }
8829 \int_compare:nNf { #2 } > \c@iRow
8830 {
8831     \int_compare:nNf { #3 } > \c@jCol
8832     {
8833         \@@_qpoint:n { row - #2 }
8834         \dim_set_eq:NN \l_tmpb_dim \pgf@y
8835         \@@_qpoint:n { col - #3 }
8836         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8837         \@@_qpoint:n
8838         { row - \int_eval:n { 1 + \int_min:nn \c@iRow { #4 } } }
8839         \dim_set_eq:NN \l_tmpa_dim \pgf@y
8840         \@@_qpoint:n
8841         { col - \int_eval:n { 1 + \int_min:nn \c@jCol { #5 } } }
8842         \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8843         \pgfpathrectanglecorners
8844         { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8845         { \pgfpoint \pgf@x \l_tmpa_dim }
8846         \dim_compare:nNfTF \l_@@_rounded_corners_dim = \c_zero_dim
8847         \pgfusepathqstroke
8848         { \pgfusepath { stroke } }
8849     }
8850 }
8851 \group_end:
8852 }

```

Here is the set of keys for the command `\@@_stroke_block:nnnnn`.

```

8853 \keys_define:nn { nicematrix / BlockStroke }
8854 {
8855     color .tl_set:N = \l_@@_draw_tl ,
8856     draw .code:n =
8857     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8858     draw .default:n = default ,
8859     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8860     rounded-corners .default:n = 4 pt ,
8861     rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The key `line-width` is now deprecated.

```

8862     line-width .dim_set:N = \l_@@_line_width_dim % deprecated
8863 }

```

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 second is the color. The other arguments are the position of the block: *imin*, *jmin*, *imax* and *jmax*.

```

8864 \cs_new_protected:Npn \@@_vlines_block:nnnnnn #1 #2 #3 #4 #5 #6
8865 {
8866     \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).

```

8867     \dim_set:Nn \arrayrulewidth { #1 }
8868     \pgfsetlinewidth \arrayrulewidth
8869     \@@_set_CTarc:n { #2 }
8870     \CT@arc@

```

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).

```

8871 \seq_set_filter:Nn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
8872 {
8873   \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #3 }
8874   ||
8875   \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #4 }
8876   ||
8877   \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #5 }
8878   ||
8879   \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #6 }
8880 }
8881 \bool_if:NT \l_@@_fix_vertex_bool
8882 {
8883   \int_compare:nNnT { #4 } > 1
8884   {
8885     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8886     {
8887       { 1 }
8888       { 1 }
8889       { \int_use:N \c@iRow }
8890       { \int_eval:n { #4 -1 } }
8891       { }
8892     }
8893   }
8894   \int_compare:nNnT { #6 } < \c@jCol
8895   {
8896     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8897     {
8898       { 1 }
8899       { \int_eval:n { #6 + 1 } }
8900       { \int_use:N \c@iRow }
8901       { \int_use:N \c@jCol }
8902       { }
8903     }
8904   }
8905 }
8906 \int_set:Nn \l_@@_start_int { #3 }
8907 \int_set:Nn \l_@@_end_int { #5 }
8908 \int_step_inline:nnn { #4 } { #6 + 1 }
8909 {
8910   \int_set:Nn \l_@@_position_int { ##1 }
8911   \socket_use:n { nicematrix / draw-vrule }
8912 }
8913 \group_end:
8914 }

```

The command `\@@_hlines_block:nnnnnn` is used only once: in `\@@_Block_v:nnnnnn` which actually draws the block.

```

8915 \cs_new_protected:Npn \@@_hlines_block:nnnnnn #1 #2 #3 #4 #5 #6
8916 {
8917   \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).

```

8918 \dim_set:Nn \arrayrulewidth { #1 }
8919 \pgfsetlinewidth \arrayrulewidth
8920 \@@_set_CTarc:n { #2 }
8921 \CT@arc@

```


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).

```

8922 \seq_set_filter:NnN \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
8923 {
8924   \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #3 }
8925   ||
8926   \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #4 }
8927   ||
8928   \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #5 }
8929   ||
8930   \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #6 }
8931 }
8932 \bool_if:NT \l_@@_fix_vertex_bool
8933 {
8934   \int_compare:nNnT { #3 } > 1
8935   {
8936     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8937     {
8938       { 1 }
8939       { 1 }
8940       { \int_eval:n { #3 - 1 } }
8941       { \int_use:N \c@jCol }
8942       { }
8943     }
8944   }
8945   \int_compare:nNnT { #5 } < \c@iRow
8946   {
8947     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8948     {
8949       { \int_eval:n { #5 + 1 } }
8950       { 1 }
8951       { \int_use:N \c@iRow }
8952       { \int_use:N \c@jCol }
8953       { }
8954     }
8955   }
8956 }
8957 \int_set:Nn \l_@@_start_int { #4 }
8958 \int_set:Nn \l_@@_end_int { #6 }
8959 \int_step_inline:nnn { #3 } { #5 + 1 }
8960 {
8961   \int_set:Nn \l_@@_position_int { ##1 }
8962   \socket_use:n { nicematrix / draw-hrule }
8963 }
8964 \group_end:
8965 }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke.

```

8966 \cs_new_protected:Npn \@@_stroke_borders_block:nnnnn #1 #2 #3 #4 #5
8967 {
8968   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8969   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8970
8971   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8972   { \@@_error:n { borders~forbidden } }
8973   {
8974     \tl_clear_new:N \l_@@_borders_tikz_tl
8975     \keys_set:no { nicematrix / OnlyForTikzInBorders } \l_@@_borders_clist
8976     \tl_set:Nn \l_@@_tmpc_tl { #2 }
8977     \tl_set:Nn \l_@@_tmpd_tl { #3 }
8978     \tl_set:Ne \l_@@_tmpa_tl { \int_eval:n { #4 + 1 } }

```

```

8979     \tl_set:Np \l_tmpb_tl { \int_eval:n { #5 + 1 } }
8980     \@@_stroke_borders_block_i:
8981   }
8982 }

8983 \AtBeginDocument
8984 {
8985   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8986   {
8987     \c_@@_pgfortikzpicture_tl
8988     \@@_stroke_borders_block_ii:
8989     \c_@@_endpgfortikzpicture_tl
8990   }
8991 }

8992 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8993 {
8994   \pgfrememberpicturepositiononpagetrue
8995   \pgf@relevantforpicturesizefalse
8996   \CT@arc@
8997   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

If there is one specification of borders (right, left, top or bottom), we will raise the flag `\l_tmpa_bool` (and, if there isn't any specification of border, we will draw all the borders).

```

8998   \bool_set_false:N \l_tmpa_bool
8999   \clist_if_in:NnT \l_@@_borders_clist { right }
9000   {
9001     \@@_stroke_vertical:n \l_tmpb_tl
9002     \bool_set_true:N \l_tmpa_bool
9003   }
9004   \clist_if_in:NnT \l_@@_borders_clist { left }
9005   {
9006     \@@_stroke_vertical:n \l_@@_tmpd_tl
9007     \bool_set_true:N \l_tmpa_bool
9008   }
9009   \clist_if_in:NnT \l_@@_borders_clist { bottom }
9010   {
9011     \@@_stroke_horizontal:n \l_tmpa_tl
9012     \bool_set_true:N \l_tmpa_bool
9013   }
9014   \clist_if_in:NnT \l_@@_borders_clist { top }
9015   {
9016     \@@_stroke_horizontal:n \l_@@_tmpc_tl
9017     \bool_set_true:N \l_tmpa_bool
9018   }
9019   \bool_if:NF \l_tmpa_bool
9020   {
9021     \@@_stroke_vertical:n \l_tmpb_tl
9022     \@@_stroke_vertical:n \l_@@_tmpd_tl
9023     \@@_stroke_horizontal:n \l_tmpa_tl
9024     \@@_stroke_horizontal:n \l_@@_tmpc_tl
9025   }
9026 }

9027 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
9028 {
9029   tikz .code:n =
9030     \cs_if_exist:NTF \tikzpicture
9031     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
9032     { \@@_error:n { tikz~in~borders~without~tikz } } ,
9033   tikz .value_required:n = true ,
9034   dashed .meta:n = { tikz = dashed } ,
9035   top .code:n = ,
9036   bottom .code:n = ,
9037   left .code:n = ,

```

```

9038     right .code:n = ,
9039     all .code:n = ,
9040     unknown .code:n = \@@_error:n { bad~border }
9041 }

```

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).

```

9042 \cs_new_protected:Npn \@@_stroke_vertical:n #1
9043 {
9044   \@@_qpoint:n \l_@@_tmpc_tl
9045   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
9046   \@@_qpoint:n \l_tmpa_tl
9047   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
9048   \@@_qpoint:n { #1 }
9049   \tl_if_empty:NTF \l_@@_borders_tikz_tl
9050   {
9051     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
9052     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
9053     \pgfusepathqstroke
9054   }
9055   {
9056     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
9057     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
9058   }
9059 }

```

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).

```

9060 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
9061 {
9062   \@@_qpoint:n \l_@@_tmpd_tl
9063   \clist_if_in:NnTF \l_@@_borders_clist { left }
9064   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
9065   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
9066   \@@_qpoint:n \l_tmpb_tl
9067   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
9068   \@@_qpoint:n { #1 }
9069   \tl_if_empty:NTF \l_@@_borders_tikz_tl
9070   {
9071     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
9072     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9073     \pgfusepathqstroke
9074   }
9075   {
9076     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
9077     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
9078   }
9079 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

9080 \keys_define:nn { nicematrix / BlockBorders }
9081 {
9082   borders .clist_set:N = \l_@@_borders_clist ,
9083   borders .default:n = all ,
9084   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
9085   rounded-corners .default:n = 4 pt ,
9086   rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The following key is deprecated.

```

9087   line-width .dim_set:N = \l_@@_line_width_dim % deprecated
9088 }

```

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.

```

9089 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
9090 {
9091   \begin { tikzpicture }
9092   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because `#5` is a list of lists.

```

9093   \clist_map_inline:nn { #1 }
9094   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

9095     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
9096     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
9097     (
9098       [
9099         xshift = \dim_use:N \l_@@_xoffset_dim ,
9100         yshift = - \dim_use:N \l_@@_yoffset_dim
9101       ]
9102       #2 -| #3
9103     )
9104     rectangle
9105     (
9106       [
9107         xshift = - \dim_use:N \l_@@_xoffset_dim ,
9108         yshift = \dim_use:N \l_@@_yoffset_dim
9109       ]
9110       \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
9111     ) ;
9112   }
9113   \end { tikzpicture }
9114 }
9115 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

```

```

9116 \keys_define:nn { nicematrix / SpecialOffset }
9117 {
9118   xoffset .dim_set:N = \l_@@_xoffset_dim ,
9119   yoffset .dim_set:N = \l_@@_yoffset_dim ,
9120   offset .meta:n = { xoffset = #1 , yoffset = #1 }
9121 }

```

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.

```

9122 \cs_new_protected:Npn \@@_NullBlock:
9123 { \@@_collect_options:n { \@@_NullBlock_i: } }
9124 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
9125 { }

```

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.

```

9126 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
9127 {
9128   \tl_gput_right:Ne \g_@@_pre_code_before_tl
9129   {

```

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).

```

9130     \@@_subcellcolor:nnnnnnn
9131     {
9132         \tl_if_blank:nTF { #1 }
9133         { { \exp_not:n { #2 } } }
9134         { [ #1 ] { \exp_not:n { #2 } } }
9135     }
9136     { \int_use:N \l_@@_first_row_int } % first row of the block
9137     { \int_use:N \l_@@_first_col_int } % first column of the block
9138     { \int_use:N \l_@@_last_row_int } % last row of the block
9139     { \int_use:N \l_@@_last_col_int } % last column of the block
9140     { \int_use:N \l_@@_split_int }
9141     { \int_use:N \l_@@_split_i_int }
9142 }
9143 \ignorespaces
9144 }
9145 \cs_new_protected:Npn \@@_subcellcolor:nnnnnnn #1 #2 #3 #4 #5 #6 #7
9146 {
9147     \@@_color_opacity: #1
9148     \pgfpicture
9149     \pgf@relevantforpicturesizefalse
9150     \@@_qpoint:n { col - #3 }
9151     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
9152     \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
9153     \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
9154     \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
9155     \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
9156     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
9157     \@@_qpoint:n { row - #2 }
9158     \pgfpathrectanglecorners
9159     { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } \l_@@_tmpc_dim }
9160     { \pgfpoint \l_tmpb_dim \pgf@y }
9161     \pgfusepathqfill
9162     \endpgfpicture
9163 }

```

27 Automatic arrays

We will extract some keys and pass the other keys to the environment {NiceArrayWithDelims}.

```

9164 \keys_define:nn { nicematrix / Auto }
9165 {
9166     columns-type .tl_set:N = \l_@@_columns_type_tl ,
9167     columns-type .value_required:n = true ,
9168     l .meta:n = { columns-type = l } ,
9169     r .meta:n = { columns-type = r } ,
9170     c .meta:n = { columns-type = c } ,
9171     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9172     delimiters / color .value_required:n = true ,
9173     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
9174     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
9175     delimiters .value_required:n = true ,
9176     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
9177     rounded-corners .default:n = 4 pt
9178 }
9179 \NewDocumentCommand \AutoNiceMatrixWithDelims
9180 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
9181 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

```

```

9182 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
9183 {

```

The group is for the protection of the keys.

```

9184 \group_begin:
9185 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
9186 \use:e
9187 {
9188   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
9189   { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
9190   [ \exp_not:o \l_tmpa_tl ]
9191 }
9192 \int_if_zero:nT \l_@@_first_row_int
9193 {
9194   \int_if_zero:nT \l_@@_first_col_int { & }
9195   \prg_replicate:nn { #4 - 1 } { & }
9196   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9197 }
9198 \prg_replicate:nn { #3 }
9199 {
9200   \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).

```

9201   \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
9202   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9203 }
9204 \int_compare:nNnT \l_@@_last_row_int > { -2 }
9205 {
9206   \int_if_zero:nT \l_@@_first_col_int { & }
9207   \prg_replicate:nn { #4 - 1 } { & }
9208   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9209 }
9210 \end { NiceArrayWithDelims }
9211 \group_end:
9212 }
9213 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
9214 {
9215   \cs_set_protected:cpn { #1 AutoNiceMatrix }
9216   {
9217     \bool_gset_true:N \g_@@_delims_bool
9218     \str_gset:Ne \g_@@_name_env_str { #1 AutoNiceMatrix }
9219     \AutoNiceMatrixWithDelims { #2 } { #3 }
9220   }
9221 }

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

9222 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
9223 {
9224   \group_begin:
9225   \bool_gset_false:N \g_@@_delims_bool
9226   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
9227   \group_end:
9228 }

```

28 The redefinition of the command \dotfill

```

9229 \cs_set_eq:NN \@@_old_dotfill: \dotfill
9230 \cs_new_protected:Npn \@@_dotfill:
9231 {

```

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}`).

```

9232 \@@_old_dotfill:
9233 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
9234 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

9235 \cs_new_protected:Npn \@@_dotfill_i:
9236 {
9237   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim
9238     { \@@_old_dotfill: }
9239 }

```

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.

```

9240 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
9241 {
9242   \tl_gput_right:Ne \g_@@_rules_tl
9243   {
9244     \@@_draw_diagbox:nnnnnn
9245     { \int_use:N \c@iRow }
9246     { \int_use:N \c@jCol }
9247     { \int_use:N \c@iRow }
9248     { \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.

```

9249     { \g_@@_row_style_tl \exp_not:n { #1 } }
9250     { \g_@@_row_style_tl \exp_not:n { #2 } }
9251   }

```

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.

```

9252   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
9253   {
9254     { \int_use:N \c@iRow }
9255     { \int_use:N \c@jCol }
9256     { \int_use:N \c@iRow }
9257     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

9258     { }
9259   }
9260 }

```

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.

```

9261 \cs_new_protected:Npn \@@_draw_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
9262 {

```

We *must* use an environment `{pgfscope}` and not a simple group of TeX.

```

9263 \begin { pgfscope }
9264 \@@_qpoint:n { row - #1 }
9265 \dim_set_eq:NN \l_tmpa_dim \pgf@y
9266 \@@_qpoint:n { col - #2 }
9267 \dim_set_eq:NN \l_tmpb_dim \pgf@x
9268 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
9269 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
9270 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
9271 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
9272 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
9273 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
9274 {

```

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.

```

9275 \CT@arc@
9276 \pgfsetroundcap
9277 \pgfusepathqstroke
9278 }
9279 \pgfset { inner~sep = 1 pt }
9280 \pgfscope
9281 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
9282 \pgfnode { rectangle } { south~west }
9283 {
9284 \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

9285 \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
9286 \end { minipage }
9287 }
9288 { }
9289 { \pgfusepath { discard } } % mandatory
9290 \endpgfscope
9291 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
9292 \pgfnode { rectangle } { north~east }
9293 {
9294 \begin { minipage } { 20 cm }
9295 \raggedleft
9296 \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
9297 \end { minipage }
9298 }
9299 { }
9300 { \pgfusepath { discard } } % mandatory
9301 \end { pgfscope }
9302 }

```

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. 90.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

9303 \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 `\`.

```

9304 \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`.

```

9305 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
9306 {
9307   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
9308   \@@_CodeAfter_iv:n
9309 }

```

We catch the argument of the command `\end` (in `#1`).

```

9310 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
9311 {

```

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.

```

9312   \str_if_eq:eeTF \currenvir { #1 }
9313   { \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`.

```

9314   {
9315     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
9316     \@@_CodeAfter_ii:n
9317   }
9318 }

```

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.

```

9319 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
9320 {
9321   \pgfpicture
9322   \pgfrememberpicturepositiononpagetrue
9323   \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.

```

9324   \@@_qpoint:n { row - 1 }
9325   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
9326   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
9327   \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).

```

9328   \bool_if:nTF { #3 }
9329   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
9330   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
9331   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
9332   {
9333     \cs_if_exist:cT
9334     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }

```

```

9335     {
9336         \pgfpointanchor
9337         { \@@_env: - ##1 - #2 }
9338         { \bool_if:nTF { #3 } { west } { east } }
9339         \dim_set:Nn \l_tmpa_dim
9340         {
9341             \bool_if:nTF { #3 }
9342             \dim_min:nn
9343             \dim_max:nn
9344             \l_tmpa_dim
9345             \pgf@x
9346         }
9347     }
9348 }

```

Now we can put the delimiter with a node of PGF.

```

9349 \pgfset { inner~sep = \c_zero_dim }
9350 \dim_zero:N \nulldelimiterspace
9351 \pgftransformshift
9352 {
9353     \pgfpoint
9354     \l_tmpa_dim
9355     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
9356 }
9357 \pgfnode
9358 { rectangle }
9359 { \bool_if:nTF { #3 } { east } { west } }
9360 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

9361     \nullfont
9362     $ % $
9363     \@@_color:o \l_@@_delimiters_color_tl
9364     \bool_if:nTF { #3 } { \left #1 } { \left . }
9365     \vcenter
9366     {
9367         \nullfont
9368         \hrule \@height
9369             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
9370             \@depth \c_zero_dim
9371             \@width \c_zero_dim
9372     }
9373     \bool_if:nTF { #3 } { \right . } { \right #1 }
9374     $ % $
9375 }
9376 { }
9377 { }
9378 \endpgfpicture
9379 }

```

32 The command \SubMatrix

```

9380 \keys_define:nn { nicematrix / sub-matrix }
9381 {
9382     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
9383     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
9384     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
9385     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
9386     xshift .value_required:n = true ,
9387     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9388     delimiters / color .value_required:n = true ,

```

```

9389     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
9390     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9391     hlines .default:n = all ,
9392     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9393     vlines .default:n = all ,
9394     hvlines .meta:n = { hlines, vlines } ,
9395     hvlines .value_forbidden:n = true
9396   }
9397 \keys_define:nn { nicematrix }
9398 {
9399     SubMatrix .inherit:n = nicematrix / sub-matrix ,
9400     NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9401     pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9402     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9403 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

9404 \keys_define:nn { nicematrix / SubMatrix }
9405 {
9406     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9407     delimiters / color .value_required:n = true ,
9408     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9409     hlines .default:n = all ,
9410     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9411     vlines .default:n = all ,
9412     hvlines .meta:n = { hlines, vlines } ,
9413     hvlines .value_forbidden:n = true ,
9414     name .code:n =
9415         \tl_if_empty:nTF { #1 }
9416         { \@@_error:n { Invalid-name } }
9417         {
9418             \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
9419             {
9420                 \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
9421                 { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
9422                 {
9423                     \str_set:Nn \l_@@_submatrix_name_str { #1 }
9424                     \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
9425                 }
9426             }
9427             { \@@_error:n { Invalid-name } }
9428         } ,
9429     name .value_required:n = true ,
9430     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
9431     rules .value_required:n = true ,
9432     code .tl_set:N = \l_@@_code_tl ,
9433     code .value_required:n = true ,
9434     unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
9435 }

9436 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
9437 {
9438     \tl_gput_right:Ne \g_@@_pre_code_after_tl
9439     {
9440         \SubMatrix { #1 } { #2 } { #3 } { #4 }
9441         [
9442             delimiters / color = \l_@@_delimiters_color_tl ,
9443             hlines = \l_@@_submatrix_hlines_clist ,
9444             vlines = \l_@@_submatrix_vlines_clist ,
9445             extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
9446             left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
9447             right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,

```

```

9448         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
9449         #5
9450     ]
9451 }
9452 \@@_SubMatrix_in_code_before_i { #2 } { #3 }
9453 \ignorespaces
9454 }
9455 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
9456 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9457 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
9458 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
9459 {
9460     \seq_gput_right:Ne \g_@@_submatrix_seq
9461     {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

9462     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
9463     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
9464     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
9465     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
9466 }
9467 }

```

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).

```

9468 \NewDocumentCommand \@@_compute_i_j:nn
9469 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9470 { \@@_compute_i_j:nnnn #1 #2 }
9471 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
9472 {
9473     \def \l_@@_first_i_tl { #1 }
9474     \def \l_@@_first_j_tl { #2 }
9475     \def \l_@@_last_i_tl { #3 }
9476     \def \l_@@_last_j_tl { #4 }
9477     \tl_if_eq:NnT \l_@@_first_i_tl { last }
9478     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
9479     \tl_if_eq:NnT \l_@@_first_j_tl { last }
9480     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
9481     \tl_if_eq:NnT \l_@@_last_i_tl { last }
9482     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
9483     \tl_if_eq:NnT \l_@@_last_j_tl { last }
9484     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
9485 }

```

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`.

```

9486 \AtBeginDocument
9487 {
9488   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m O { } E { _ ^ } { { } { } } }
9489   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
9490     { \@@_sub_matrix:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
9491 }
9492 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
9493 {
9494   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

9495 \@@_compute_i_j:nn { #2 } { #3 }
9496 \int_compare:nNt \l_@@_first_i_tl = \l_@@_last_i_tl
9497 { \def \arraystretch { 1 } }
9498 \bool_lazy_or:nnTF
9499 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9500 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9501 { \@@_error:nn { Construct-too-large } { \SubMatrix } }
9502 {
9503   \str_clear_new:N \l_@@_submatrix_name_str
9504   \keys_set:nn { nicematrix / SubMatrix } { #5 }
9505   \pgfpicture
9506   \pgfrememberpicturepositiononpagetrue
9507   \pgf@relevantforpicturesizefalse
9508   \pgfset { inner~sep = \c_zero_dim }
9509   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9510   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curryfication.

```

9511 \bool_if:Ntf \l_@@_submatrix_slim_bool
9512 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
9513 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
9514 {
9515   \cs_if_exist:cT
9516   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9517   {
9518     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9519     \dim_compare:nNt \pgf@x < \l_@@_x_initial_dim
9520     { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9521   }
9522   \cs_if_exist:cT
9523   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9524   {
9525     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9526     \dim_compare:nNt \pgf@x > \l_@@_x_final_dim
9527     { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9528   }
9529 }
9530 \dim_compare:nNtTF \l_@@_x_initial_dim = \c_max_dim
9531 { \@@_impossible_delimiter:n { left } }
9532 {
9533   \dim_compare:nNtTF \l_@@_x_final_dim = { - \c_max_dim }
9534   { \@@_impossible_delimiter:n { right } }
9535   { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
9536 }
9537 \endpgfpicture
9538 }
9539 \group_end:
9540 \ignorespaces
9541 }

```

The argument of the following command will be provided by curryfication.

```

9542 \cs_new_protected:Npn \@@_impossible_delimiter:n #1
9543 {
9544   \bool_if:NTF \l_@@_no_cell_nodes_bool
9545     { \@@_error:n { Impossible~SubMatrix~no~cell~nodes } }
9546     { \@@_error:nn { Impossible~SubMatrix } { #1 } }
9547 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

9548 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
9549 {
9550   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
9551   \dim_set:Nn \l_@@_y_initial_dim
9552     {
9553     \fp_to_dim:n
9554       {
9555         \pgf@y
9556         + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
9557       }
9558     }
9559   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
9560   \dim_set:Nn \l_@@_y_final_dim
9561     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
9562   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
9563     {
9564       \cs_if_exist:cT
9565         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
9566         {
9567           \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
9568           \dim_set:Nn \l_@@_y_initial_dim
9569             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
9570         }
9571       \cs_if_exist:cT
9572         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
9573         {
9574           \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
9575           \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
9576             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
9577         }
9578     }
9579   \dim_set:Nn \l_tmpa_dim
9580     {
9581       \l_@@_y_initial_dim - \l_@@_y_final_dim +
9582       \l_@@_submatrix_extra_height_dim - \arrayrulewidth
9583     }
9584   \dim_zero:N \nullldelimiterspace

```

We will draw the rules in the \SubMatrix.

```

9585 \group_begin:
9586 \pgfsetlinewidth { 1.1 \arrayrulewidth }
9587 \@@_set_CTarc:o \l_@@_rules_color_tl
9588 \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`.

```

9589 \seq_map_inline:Nn \g_@@_cols_vlism_seq
9590 {
9591   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
9592   {
9593     \int_compare:nNnT
9594       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
9595     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

9596         \@@_qpoint:n { col - ##1 }
9597         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9598         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9599         \pgfusepathqstroke
9600     }
9601 }
9602 }

```

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.

```

9603 \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
9604 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
9605 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
9606 {
9607     \bool_lazy_and:nnTF
9608     { \int_compare_p:nNn { ##1 } > \c_zero_int }
9609     {
9610         \int_compare_p:nNn
9611         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
9612     {
9613         \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
9614         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9615         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9616         \pgfusepathqstroke
9617     }
9618     { \@@_error:mn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
9619 }

```

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.

```

9620 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
9621 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
9622 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
9623 {
9624     \bool_lazy_and:nnTF
9625     { \int_compare_p:nNn { ##1 } > \c_zero_int }
9626     {
9627         \int_compare_p:nNn
9628         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
9629     {
9630         \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

9631 \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

9632 \dim_set:Nn \l_tmpa_dim
9633 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9634 \str_case:nn { #1 }
9635 {
9636     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9637     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
9638     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9639     }
9640     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

9641 \dim_set:Nn \l_tmpb_dim
9642 { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9643 \str_case:nn { #2 }
9644 {
9645     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9646     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }

```

```

9647         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9648     }
9649     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9650     \pgfusepath{stroke}
9651     \group_end:
9652 }
9653 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
9654 }

```

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).

```

9655 \str_if_empty:NF \l_@@_submatrix_name_str
9656 {
9657     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
9658     \l_@@_x_initial_dim \l_@@_y_initial_dim
9659     \l_@@_x_final_dim \l_@@_y_final_dim
9660 }
9661 \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`.

```

9662 \begin { pgfscope }
9663 \pgftransformshift
9664 {
9665     \pgfpoint
9666     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9667     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9668 }
9669 \str_if_empty:NTF \l_@@_submatrix_name_str
9670 { \@@_node_left:nn #1 { } }
9671 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9672 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9673 \pgftransformshift
9674 {
9675     \pgfpoint
9676     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9677     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9678 }
9679 \str_if_empty:NTF \l_@@_submatrix_name_str
9680 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9681 {
9682     \@@_node_right:nnnn #2
9683     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9684 }

```

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`.

```

9685 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9686 \flag_clear_new:N \l_@@_code_flag
9687 \l_@@_code_tl
9688 }

```

In the key code of the command `\SubMatrix` there may be TikZ instructions. We want that, in these instructions, the *i* and *j* in specifications of nodes of the forms *i-j*, *row-i*, *col-j* and *i-|j* refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

9689 \cs_set_eq:NN \@@_old_pgfpntanchor: \pgfpointanchor

```


The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of TikZ nodes which are computed in an expandable way.

The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by curryfication.

```
9690 \cs_new:Npn \@@_pgfpointanchor:n #1
9691 { \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`).

```
9692 \cs_new:Npn \@@_pgfpointanchor_i:n #1
9693 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }
9694 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9695 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
9696 \str_if_empty:nTF { #1 }
```

First, when the name of the name begins with `\tikz@pp@name`.

```
9697 { \@@_pgfpointanchor_iv:w #2 }
```

And now, when there is no `\tikz@pp@name`.

```
9698 { \@@_pgfpointanchor_ii:n { #1 } }
9699 }
```

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`).

```
9700 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
9701 { \@@_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`.

```
9702 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }
```

```
9703 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
9704 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
9705 \str_if_empty:nTF { #2 }
```

First the case where the argument does *not* contain an hyphen.

```
9706 { \@@_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`).

```
9707 { \@@_pgfpointanchor_iii:w { #1 } #2 }
9708 }
```

The following function is for the case when the name contains an hyphen.

```
9709 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9710 {
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
9711 \@@_env:
9712 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9713 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9714 }
```

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.

```

9715 \tl_const:Nn \c_@@_integers_alist_tl
9716 {
9717   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9718   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9719   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9720   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9721 }

9722 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9723 {

```

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`.

```

9724   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
9725   {
9726     \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9727   \@@_env: -
9728   \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9729     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9730     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9731   }
9732   {
9733     \str_if_eq:eeTF { #1 } { last }
9734     {
9735       \flag_raise:N \l_@@_code_flag
9736       \@@_env: -
9737       \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9738         { \int_eval:n { \l_@@_last_i_tl + 1 } }
9739         { \int_eval:n { \l_@@_last_j_tl + 1 } }
9740     }
9741     { #1 }
9742   }
9743 }

```

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`).

```

9744 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9745 {
9746   \pgfnode
9747     { rectangle }
9748     { east }
9749     {
9750       \nullfont
9751       $ % $
9752       \@@_color:o \l_@@_delimiters_color_tl
9753       \left #1
9754       \vcenter
9755       {
9756         \nullfont
9757         \hrule \@height \l_tmpa_dim
9758         \@depth \c_zero_dim

```

```

9759         \@width \c_zero_dim
9760     }
9761     \right .
9762     $ % $
9763 }
9764 { #2 }
9765 { }
9766 }

```

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.

```

9767 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9768 {
9769     \pgfnode
9770     { rectangle }
9771     { west }
9772     {
9773         \nullfont
9774         $ % $
9775         \colorlet { current-color } { . }
9776         \@@_color:o \l_@@_delimiters_color_tl
9777         \left .
9778         \vcenter
9779         {
9780             \nullfont
9781             \hrule \@height \l_tmpa_dim
9782                 \@depth \c_zero_dim
9783                 \@width \c_zero_dim
9784         }
9785         \right #1
9786         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9787         ^ { \color { current-color } \smash { #4 } }
9788         $ % $
9789     }
9790     { #2 }
9791     { }
9792 }

```

33 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9793 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9794 {
9795     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9796     \ignorespaces
9797 }
9798 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9799 {
9800     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9801     \ignorespaces
9802 }
9803 \keys_define:nn { nicematrix / Brace }
9804 {
9805     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9806     left-shorten .value_forbidden:n = true ,
9807     right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,

```

```

9808 right-shorten .value_forbidden:n = true ,
9809 shorten .meta:n = { left-shorten , right-shorten } ,
9810 shorten .value_forbidden:n = true ,
9811 yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9812 color .tl_set:N = \l_tmpa_tl ,
9813 color .value_required:n = true ,
9814 unknown .code:n =
9815   \@@_unknown_key:nn
9816   { nicematrix / Brace }
9817   { Unknown-key-for-Brace }
9818 }

```

#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`.

```

9819 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9820 {
9821   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9822   \@@_compute_i_j:nn { #1 } { #2 }
9823   \bool_lazy_or:nnTF
9824     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9825     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9826     {
9827       \str_if_eq:eeTF { #5 } { under }
9828       { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9829       { \@@_error:nn { Construct-too-large } { \OverBrace } }
9830     }
9831   {
9832     \tl_clear:N \l_tmpa_tl
9833     \keys_set:nn { nicematrix / Brace } { #4 }
9834     \tl_if_empty:NF \l_tmpa_tl { \color \l_tmpa_tl }
9835     \pgfpicture
9836     \pgfrememberpicturerepositiononpagetrue
9837     \pgf@relevantforpicturesizefalse
9838     \bool_if:NT \l_@@_brace_left_shorten_bool
9839     {
9840       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9841       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9842       {
9843         \cs_if_exist:cT
9844           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9845           {
9846             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9847
9848             \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9849             { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9850           }
9851       }
9852     }
9853   \bool_lazy_or:nnT
9854     { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9855     { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
9856     {
9857       \@@_qpoint:n { col - \l_@@_first_j_tl }
9858       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9859     }
9860   \bool_if:NT \l_@@_brace_right_shorten_bool
9861   {
9862     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9863     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9864     {
9865       \cs_if_exist:cT

```

```

9866         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9867         {
9868             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9869             \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9870             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9871         }
9872     }
9873 }
9874 \bool_lazy_or:nnT
9875 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9876 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
9877 {
9878     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9879     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9880 }
9881 \pgfset { inner~sep = \c_zero_dim }
9882 \str_if_eq:eeTF { #5 } { under }
9883 { \@@_underbrace_i:n { #3 } }
9884 { \@@_overbrace_i:n { #3 } }
9885 \endpgfpicture
9886 }
9887 \group_end:
9888 }

```

The argument is the text to put above the brace.

```

9889 \cs_new_protected:Npn \@@_overbrace_i:n #1
9890 {
9891     \@@_qpoint:n { row - \l_@@_first_i_tl }
9892     \pgftransformshift
9893     {
9894         \pgfpoint
9895         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9896         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9897     }
9898     \pgfnode
9899     { rectangle }
9900     { south }
9901     {
9902         \vtop
9903         {
9904             \group_begin:
9905             \everycr { }
9906             \halign
9907             {
9908                 \hfil ## \hfil \crcr
9909                 \bool_if:NTF \l_@@_tabular_bool
9910                 { \begin { tabular } { c } #1 \end { tabular } }
9911                 { $ \begin { array } { c } #1 \end { array } $ }
9912                 \cr
9913                 $ % $
9914                 \overbrace
9915                 {
9916                     \hbox_to_wd:nn
9917                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9918                     { }
9919                 }
9920                 $ % $
9921                 \cr
9922             }
9923             \group_end:
9924         }
9925     }
9926 }
9927 { }

```

```
9928 }
```

The argument is the text to put under the brace.

```
9929 \cs_new_protected:Npn \@@_underbrace_i:n #1
9930 {
9931   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9932   \pgftransformshift
9933   {
9934     \pgfpoint
9935     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9936     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9937   }
9938   \pgfnode
9939   { rectangle }
9940   { north }
9941   {
9942     \group_begin:
9943     \everycr { }
9944     \vbox
9945     {
9946       \halign
9947       {
9948         \hfil ## \hfil \crcr
9949         $ % $
9950         \underbrace
9951         {
9952           \hbox_to_wd:nn
9953           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9954           { }
9955         }
9956         $ % $
9957         \cr
9958         \bool_if:NTF \l_@@_tabular_bool
9959         { \begin { tabular } { c } #1 \end { tabular } }
9960         { $ \begin { array } { c } #1 \end { array } $ }
9961         \cr
9962       }
9963     }
9964     \group_end:
9965   }
9966   { }
9967   { }
9968 }
```

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`).

```
9969 \AddToHook { package / tikz / after }
9970 {
9971   \tikzset
9972   {
9973     nicematrix / brace / .style =
9974     {
```

```

9975         decoration = { brace , raise = -0.15 em } ,
9976         decorate ,
9977     } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9978     nicematrix / mirrored-brace / .style =
9979     {
9980         nicematrix / brace ,
9981         decoration = mirror ,
9982     }
9983 }
9984 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9985 \keys_define:nn { nicematrix / Hbrace }
9986 {
9987     color .code:n = ,
9988     horizontal-label .code:n = ,
9989     horizontal-labels .code:n = ,
9990     shorten .code:n = ,
9991     shorten-start .code:n = ,
9992     shorten-end .code:n = ,
9993     shorten+ .code:n = ,
9994     shorten-start+ .code:n = ,
9995     shorten-end+ .code:n = ,
9996     shorten~+ .code:n = ,
9997     shorten-start~+ .code:n = ,
9998     shorten-end~+ .code:n = ,
9999     brace-shift .code:n = ,
10000     brace-shift+ .code:n = ,
10001     brace-shift~+ .code:n = ,
10002     unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
10003 }

```

Here we need an “fully expandable” command.

```

10004 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
10005 {
10006     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
10007     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
10008     { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
10009 }

```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

10010 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
10011 {
10012     \int_compare:nNnTF \c@iRow < { 2 }
10013     {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

10014     \str_if_eq:nnTF { #2 } { * }
10015     {
10016         \bool_set_true:N \l_@@_nullify_dots_bool
10017         \Ldots
10018         [
10019             line-style = nicematrix / brace ,
10020             #1 ,
10021             up =
10022             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10023         ]
10024     }

```

```

10025     {
10026         \Hdotsfor
10027         [
10028             line-style = nicematrix / brace ,
10029             #1 ,
10030             up =
10031             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10032         ]
10033         { #2 }
10034     }
10035 }
10036 {
10037     \str_if_eq:nnTF { #2 } { * }
10038     {
10039         \bool_set_true:N \l_@@_nullify_dots_bool
10040         \Ldots
10041         [
10042             line-style = nicematrix / mirrored-brace ,
10043             #1 ,
10044             down =
10045             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10046         ]
10047     }
10048     {
10049         \Hdotsfor
10050         [
10051             line-style = nicematrix / mirrored-brace ,
10052             #1 ,
10053             down =
10054             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10055         ]
10056         { #2 }
10057     }
10058 }
10059 \keys_set:nn { nicematrix / Hbrace } { #1 }
10060 }

```

```

10061 \NewDocumentCommand { \@@_Vbrace } { 0 { } m m }
10062 {
10063     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
10064     { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
10065     { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
10066 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

10067 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
10068 {
10069     \int_compare:nNnTF \c@jCol < { 2 }
10070     {
10071         \str_if_eq:nnTF { #2 } { * }
10072         {
10073             \bool_set_true:N \l_@@_nullify_dots_bool
10074             \Vdots
10075             [
10076                 Vbrace ,
10077                 line-style = nicematrix / mirrored-brace ,
10078                 #1 ,
10079                 down =
10080                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10081             ]
10082         }
10083     }

```



```

10084         \Vdotsfor
10085         [
10086             Vbrace ,
10087             line-style = nicematrix / mirrored-brace ,
10088             #1 ,
10089             down =
10090                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10091         ]
10092     { #2 }
10093 }
10094 }
10095 {
10096     \str_if_eq:nnTF { #2 } { * }
10097     {
10098         \bool_set_true:N \l_@@_nullify_dots_bool
10099         \Vdots
10100         [
10101             Vbrace ,
10102             line-style = nicematrix / brace ,
10103             #1 ,
10104             up =
10105                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10106         ]
10107     }
10108     {
10109         \Vdotsfor
10110         [
10111             Vbrace ,
10112             line-style = nicematrix / brace ,
10113             #1 ,
10114             up =
10115                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10116         ]
10117         { #2 }
10118     }
10119 }
10120 \keys_set:nn { nicematrix / Hbrace } { #1 }
10121 }

```

35 The command TikzEveryCell

```

10122 \bool_new:N \l_@@_not_empty_bool
10123 \bool_new:N \l_@@_empty_bool
10124
10125 \keys_define:nn { nicematrix / TikzEveryCell }
10126 {
10127     not-empty .code:n =
10128         \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
10129         { \bool_set_true:N \l_@@_not_empty_bool }
10130         { \@@_error:n { detection-of-empty-cells } } ,
10131     not-empty .value_forbidden:n = true ,
10132     empty .code:n =
10133         \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
10134         { \bool_set_true:N \l_@@_empty_bool }
10135         { \@@_error:n { detection-of-empty-cells } } ,
10136     empty .value_forbidden:n = true ,
10137     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
10138 }
10139

```

```

10140
10141 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
10142 {
10143   \IfPackageLoadedTF { tikz }
10144   {
10145     \group_begin:
10146     \keys_set:nn { nicematrix / TikzEveryCell } { #1 }

```

The inner pair of braces in the following line is mandatory because, the last argument of `\@@_tikz:nnnnn` is a *list of lists* of TikZ keys.

```

10147     \tl_set:Nn \l_tmpa_tl { { #2 } }
10148     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
10149     { \@@_for_a_block:nnnnn ##1 }
10150     \@@_all_the_cells:
10151     \group_end:
10152   }
10153   { \@@_error:n { TikzEveryCell-without~tikz } }
10154 }
10155
10156
10157 \cs_new_protected:Nn \@@_all_the_cells:
10158 {
10159   \int_step_inline:nn \c@iRow
10160   {
10161     \int_step_inline:nn \c@jCol
10162     {
10163       \cs_if_exist:cF { cell - ##1 - #####1 }
10164       {
10165         \clist_if_in:Nef \l_@@_corners_cells_clist
10166         { ##1 - #####1 }
10167         {
10168           \bool_set_false:N \l_tmpa_bool
10169           \cs_if_exist:cTF
10170           { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
10171           {
10172             \bool_if:NF \l_@@_empty_bool
10173             { \bool_set_true:N \l_tmpa_bool }
10174           }
10175           {
10176             \bool_if:NF \l_@@_not_empty_bool
10177             { \bool_set_true:N \l_tmpa_bool }
10178           }
10179           \bool_if:NT \l_tmpa_bool
10180           {
10181             \@@_block_tikz:nnnnn
10182             \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
10183           }
10184         }
10185       }
10186     }
10187   }
10188 }
10189
10190 \cs_new_protected:Nn \@@_for_a_block:nnnnn
10191 {
10192   \bool_if:NF \l_@@_empty_bool
10193   {
10194     \@@_block_tikz:nnnnn
10195     \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
10196   }
10197   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
10198 }
10199
10200 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn

```

```

10201 {
10202   \int_step_inline:nnn { #1 } { #3 }
10203   {
10204     \int_step_inline:nnn { #2 } { #4 }
10205     { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
10206   }
10207 }

```

36 The key draw-trees-in-col

```

10208 \cs_new_protected:Npn \@@_draw_trees:
10209 {
10210   \bool_if:NTF \l_@@_no_cell_nodes_bool
10211   { \@@_error:n { draw-trees-with-no-cell-nodes } }
10212   \@@_draw_trees_i:
10213 }
10214 \cs_new_protected:Npn \@@_draw_trees_i:
10215 {
10216   \@@_expand_clist_hvlines:NN \g_@@_col_with_trees_clist \c@jCol
10217   \dim_zero_new:N \l_@@_em_dim
10218   \dim_set:Nn \l_@@_em_dim { 1 em }
10219   \dim_zero_new:N \l_@@_ex_dim
10220   \dim_set:Nn \l_@@_ex_dim { 1 ex }
10221   \pgfpicture
10222   \pgfrememberpicturepositiononpagetrue
10223   \pgf@relevantforpicturesizefalse
10224   \dim_compare:nNnT \l_@@_trees_line_width_dim > \c_zero_dim
10225   { \pgfsetlinewidth { \l_@@_trees_line_width_dim } }
10226   \@@_color:o \l_@@_trees_color_tl
10227   \pgfsetcornersarced
10228   { \pgfpoint \l_@@_trees_rounded_corners_dim \l_@@_trees_rounded_corners_dim }
10229   \clist_map_function:NN \g_@@_col_with_trees_clist
10230   \@@_draw_trees_in_col:n
10231   \clist_gclear:N \g_@@_col_with_trees_clist
10232   \endpgfpicture
10233 }
10234 \cs_new_protected:Npn \@@_draw_trees_in_col:n #1
10235 {

```

The argument is provided by curryfication.

```

10236   \int_compare:nNnTF { #1 } > \c@jCol
10237   { \@@_error:nn { Col-outside-tabular-in-trees } }
10238   {
10239     \int_compare:nNnTF { #1 } = \c@jCol
10240     { \@@_error:nn { Last-col-in-trees } }
10241     \@@_draw_trees_in_col_i:n
10242   }
10243   { #1 }
10244 }
10245 \cs_new_protected:Npn \@@_draw_trees_in_col_i:n #1
10246 {
10247   \int_set:Nn \l_tmpa_int { 1 }
10248   \int_step_inline:nn { \c@iRow + 1 }
10249   {
10250     \cs_if_exist:cT
10251     { pgf @ sh @ ns @ \@@_env: - ##1 - #1 }
10252     {
10253       \int_compare:nNnT { ##1 } > { \l_tmpa_int + 1 }
10254       {

```

Now, you will, potentially, draw a tree.

```

10255         \@@_draw_tree:nee
10256         { #1 }
10257         { \int_use:N \l_tmpa_int }
10258         { \int_eval:n { ##1 - 1 } }
10259     }
10260     \int_set:Nn \l_tmpa_int { ##1 }
10261 }
10262 }
10263 \int_compare:nNnT \c@iRow > \l_tmpa_int
10264 {
10265     \@@_draw_tree:nee
10266     { #1 }
10267     { \int_use:N \l_tmpa_int }
10268     { \int_eval:n { \c@iRow } }
10269 }
10270 }

```

#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.

```

10271 \cs_new_protected:Npn \@@_draw_tree:nnn #1 #2 #3
10272 {

```

\l_tmpa_dim will be the x -value of the vertical rule that we will draw.

```

10273 \pgfpointanchor { \@@_env: - #1 } { 5 }
10274 \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.

```

10275 \bool_set_false:N \l_tmpa_bool
10276 \int_step_inline:nnnn { #3 } { -1 } { #2 + 1 }
10277 {
10278     \cs_if_exist:cT
10279     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #1 + 1 } }

```

We have found the last branch to draw.

```

10280     {
10281         \pgfpointanchor
10282         { \@@_env: - ##1 - \int_eval:n { #1 + 1 } }
10283         { base~west }
10284         \pgfpathmoveto
10285         {
10286             \pgfpoint
10287             { \dim_eval:n { \pgf@x - 0.7 \l_@@_ex_dim } }
10288             { \dim_eval:n { \pgf@y + 0.25 \l_@@_em_dim } }
10289         }
10290         \pgfpathlineto { \pgfpoint \l_tmpa_dim \pgf@y }
10291         \bool_if:NF \l_tmpa_bool
10292         {
10293             \pgfpointanchor{ \@@_env: - #2 - #1 } { south }
10294             \pgfpathlineto
10295             { \pgfpoint \l_tmpa_dim { \dim_eval:n { \pgf@y - 3pt } } }
10296             \bool_set_true:N \l_tmpa_bool
10297         }
10298         \pgfusepath { stroke }
10299     }
10300 }
10301 }
10302 \cs_generate_variant:Nn \@@_draw_tree:nnn { n e e }

```

37 The key create-blocks-in-col

```

10303 \cs_new_protected:Npn \@@_create_blocks_in_col:

```

```

10304 {
10305   \@@_expand_clist_hvlines:NN \g_@@_cbic_clist \c@jCol
10306   \clist_map_inline:Nn \g_@@_cbic_clist
10307   {
10308     \cs_set:cpn
10309     {
10310       pgf @ sh @ ns @ \@@_env:
10311       - \int_eval:n { \c@iRow + 1 } - ##1 }
10312     { rien }
10313   }
10314   \int_set:Nn \l_tmpa_int { 1 }
10315   \int_step_inline:nn { \c@iRow + 1 }
10316   {
10317     \cs_if_exist:cT
10318     { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 }
10319     {
10320       \int_compare:nNnT { #####1 } > { \l_tmpa_int + 1 }
10321       {
10322         \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
10323         {
10324           { \int_use:N \l_tmpa_int }
10325           { ##1 }
10326           { \int_eval:n { #####1 - 1 } }
10327           { ##1 }
10328           { }
10329         }
10330       }
10331       \int_set:Nn \l_tmpa_int { #####1 }
10332     }
10333   }
10334   \clist_gclear:N \g_@@_cbic_clist
10335 }

```

38 The command \ShowCellNames

When the command `\ShowCellNames` is used in the `\CodeBefore`, we want to stroke the names of the cells *after* the potential backgrounds of the cells. That's why we add the command `\@@_ShowCellNames` to the right of the command `\@@_actually_color:` which actually fill the backgrounds.

```

10336 \cs_new_protected:Npn \@@_ShowCellNamesCodeBefore
10337 { \tl_put_right:Nn \@@_actually_color: \@@_ShowCellNames } % noqa
10338
10339 \NewDocumentCommand \@@_ShowCellNames { }
10340 {
10341   \bool_if:NT \l_@@_in_code_after_bool
10342   {
10343     \pgfpicture
10344     \pgfrememberpicturepositiononpagetrue
10345     \pgf@relevantforpicturesizefalse
10346     \pgfpathrectanglecorners
10347     { \@@_qpoint:n { 1 } }
10348     { \@@_qpoint:n { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } } }
10349     \pgfsetfillopacity { 0.75 }
10350     \pgfsetfillcolor { white }
10351     \pgfusepathqfill
10352     \endpgfpicture
10353   }
10354   \dim_gzero_new:N \g_@@_tmpc_dim

```

```

10354 \dim_gzero_new:N \g_@@_tmpd_dim
10355 \dim_gzero_new:N \g_@@_tmpe_dim
10356 \int_step_inline:nn \c@iRow
10357 {
10358   \bool_if:NTF \l_@@_in_code_after_bool
10359   {
10360     \pgfpicture
10361     \pgfrememberpicturepositiononpagetrue
10362     \pgf@relevantforpicturesizefalse
10363   }
10364   { \begin { pgfpicture } }
10365   \@@_qpoint:n { row - ##1 }
10366   \dim_set_eq:NN \l_tmpa_dim \pgf@y
10367   \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
10368   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
10369   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
10370   \bool_if:NTF \l_@@_in_code_after_bool
10371   { \endpgfpicture }
10372   { \end { pgfpicture } }
10373   \int_step_inline:nn \c@jCol
10374   {
10375     \hbox_set:Nn \l_tmpa_box
10376     {
10377       \normalfont \Large \sffamily \bfseries
10378       \bool_if:NTF \l_@@_in_code_after_bool
10379       { \color { red } }
10380       { \color { red ! 50 } }
10381       ##1 - ####1
10382     }
10383     \bool_if:NTF \l_@@_in_code_after_bool
10384     {
10385       \pgfpicture
10386       \pgfrememberpicturepositiononpagetrue
10387       \pgf@relevantforpicturesizefalse
10388     }
10389     { \begin { pgfpicture } }
10390     \@@_qpoint:n { col - ####1 }
10391     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
10392     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
10393     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
10394     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
10395     \bool_if:NTF \l_@@_in_code_after_bool
10396     { \endpgfpicture }
10397     { \end { pgfpicture } }
10398     \fp_set:Nn \l_tmpa_fp
10399     {
10400       \fp_min:nn
10401       {
10402         \fp_min:nn
10403         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
10404         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
10405       }
10406       { 1.0 }
10407     }
10408     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
10409     \pgfpicture
10410     \pgfrememberpicturepositiononpagetrue
10411     \pgf@relevantforpicturesizefalse
10412     \pgftransformshift
10413     {
10414       \pgfpoint
10415       { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
10416       \g_tmpa_dim

```

```

10417         }
10418         \pgfnode
10419         { rectangle }
10420         { center }
10421         { \box_use:N \l_tmpa_box }
10422         { }
10423         { }
10424         \endpgfpicture
10425     }
10426 }
10427 }

```

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.

```

10428 \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.

```

10429 \bool_new:N \g_@@_footnote_bool
10430 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
10431 {
10432   You-have-used-the-key~' \l_keys_key_str '~when-loading-nicematrix~
10433   but~that~key~is~unknown. \\\
10434   It~will~be~ignored. \\\
10435   For~a~list~of~the~available~keys,~type~H~<return>.
10436 }
10437 {
10438   The-available-keys-are~(in~alphabetic-order):~
10439   footnote,~
10440   footnotehyper,~
10441   messages-for-Overleaf,~
10442   renew-dots~and~
10443   renew-matrix.
10444 }
10445 \keys_define:nn { nicematrix }
10446 {
10447   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
10448   renew-dots .value_forbidden:n = true ,
10449   renew-matrix .code:n = \@@_renew_matrix: ,
10450   renew-matrix .value_forbidden:n = true ,
10451   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
10452   footnote .bool_set:N = \g_@@_footnote_bool ,
10453   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
10454   unknown .code:n = \@@_error:n { Unknown~key~for~package }
10455 }
10456 \ProcessKeyOptions
10457 \@@_msg_new:nn { footnote~with~footnotehyper~package }
10458 {
10459   You-can't-use-the-option~'footnote'~because~the~package~

```

```

10460 footnotehyper~has~already~been~loaded.~
10461 If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
10462 within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10463 of~the~package~footnotehyper.\\
10464 The~package~footnote~won't~be~loaded.
10465 }
10466 \@@_msg_new:nn { footnotehyper~with~footnote~package }
10467 {
10468   You~can't~use~the~option~'footnotehyper'~because~the~package~
10469   footnote~has~already~been~loaded.~
10470   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
10471   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10472   of~the~package~footnote.\\
10473   The~package~footnotehyper~won't~be~loaded.
10474 }
10475 \bool_if:NT \g_@@_footnote_bool
10476 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10477   \IfClassLoadedTF { beamer }
10478   { \bool_set_false:N \g_@@_footnote_bool }
10479   {
10480     \IfPackageLoadedTF { footnotehyper }
10481     { \@@_error:n { footnote~with~footnotehyper~package } }
10482     { \usepackage { footnote } }
10483   }
10484 }
10485 \bool_if:NT \g_@@_footnotehyper_bool
10486 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10487   \IfClassLoadedTF { beamer }
10488   { \bool_set_false:N \g_@@_footnote_bool }
10489   {
10490     \IfPackageLoadedTF { footnote }
10491     { \@@_error:n { footnotehyper~with~footnote~package } }
10492     { \usepackage { footnotehyper } }
10493   }
10494   \bool_set_true:N \g_@@_footnote_bool
10495 }

```

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.

```

10496 \bool_new:N \l_@@_underscore_loaded_bool
10497 \IfPackageLoadedT { underscore }
10498 { \bool_set_true:N \l_@@_underscore_loaded_bool }
10499 \AtBeginDocument
10500 {
10501   \bool_if:NF \l_@@_underscore_loaded_bool
10502   {

```



```

10503     \IfPackageLoadedT { underscore }
10504     { \@@_error:n { underscore-after-nicematrix } }
10505   }
10506 }

```

41 Compatibility with threeparttable

```

10507 \AtBeginDocument
10508 {
10509   \IfPackageLoadedT { threeparttable }
10510   {
10511     \AddToHook { env / threeparttable / begin }
10512     {
10513       \TPT@hookin { NiceTabular }
10514       \TPT@hookin { NiceTabular* }
10515       \TPT@hookin { NiceTabularX }
10516     }
10517   }
10518 }

```

42 Gestion of the errors

42.1 Security in the CodeBefore and the CodeAfter

```

10519 \cs_new_protected:Npn \@@_security_font_commands:
10520 {
10521   \cs_set_eq:NN \@@_old_small: \small
10522   \cs_set_eq:NN \@@_old_footnotesize: \footnotesize
10523   \cs_set_eq:NN \@@_old_scriptsize: \scriptsize
10524   \cs_set_eq:NN \@@_old_tiny: \tiny
10525   \cs_set_eq:NN \small \@@_small:
10526   \cs_set_eq:NN \footnotesize \@@_footnotesize:
10527   \cs_set_eq:NN \scriptsize \@@_scriptsize:
10528   \cs_set_eq:NN \tiny \@@_tiny:
10529   \everyhbox
10530   {
10531     \cs_set_eq:NN \small \@@_old_small:
10532     \cs_set_eq:NN \footnotesize \@@_old_footnotesize:
10533     \cs_set_eq:NN \scriptsize \@@_old_scriptsize:
10534     \cs_set_eq:NN \tiny \@@_old_tiny:
10535   }
10536 }
10537 \cs_set_protected:Npn \@@_small: { \@@_font_command_error:N \small }
10538 \cs_set_protected:Npn \@@_footnotsize: { \@@_font_command_error:N \footnotesize }
10539 \cs_set_protected:Npn \@@_scriptsize: { \@@_font_command_error:N \scriptsize }
10540 \cs_set_protected:Npn \@@_tiny: { \@@_font_command_error:N \tiny }
10541 \cs_new_protected:Npn \@@_font_command_error:N #1
10542 { \@@_error:nn { font-command } { #1 } }
10543 \@@_msg_new:nn { font-command }
10544 {
10545   Font-command-not-allowed.\@
10546   You-can't-use-the-command-#1 directly-in-the-
10547   \bool_if:NTF \l_@@_in_code_after_bool
10548   { \token_to_str:N \CodeAfter}
10549   { \token_to_str:N \CodeBefore}.~Your-command-will-be-ignored.
10550 }

```

42.2 The 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.

```

10551 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
10552 {
10553   \str_if_eq:eeTF
10554     { \str_item:Nn \l_keys_key_str { \str_count:N \l_keys_key_str } }
10555     { + }
10556     {
10557       \str_set:Ne \l_tmpa_str
10558         { \str_range:Nnn \l_keys_key_str { 1 } { \str_count:N \l_keys_key_str - 1 } }
10559       \bool_set_false:N \l_tmpa_bool
10560       \clist_map_inline:nn { #1 }
10561         {
10562           \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10563           {
10564             \@@_error:n { key~without~+~exists }
10565             \bool_set_true:N \l_tmpa_bool
10566             \clist_map_break:
10567           }
10568         }
10569       \bool_if:NF \l_tmpa_bool
10570       {
10571         \str_set:Ne \l_keys_key_str { \tl_trim_right_spaces:V \l_tmpa_str }
10572         \@@_unknown_key_i:nn { #1 } { #2 }
10573       }
10574     }
10575   { \@@_unknown_key_i:nn { #1 } { #2 } }
10576 }

```

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.

```

10577 \cs_new_protected:Npn \@@_unknown_key_i:nn #1 #2
10578 {
10579   \str_set_eq:NN \l_tmpa_str \l_keys_key_str
10580   \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
10581   \str_set:Ne \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
10582   \bool_set_false:N \l_tmpa_bool
10583   \clist_map_inline:nn { #1 }
10584     {
10585       \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10586       {
10587         \@@_error:n { key~with~normal~form~exists }
10588         \bool_set_true:N \l_tmpa_bool
10589         \clist_map_break:
10590       }
10591     }
10592   \bool_if:NF \l_tmpa_bool
10593   {
10594     \@@_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).

```

10595   \bool_if:NF \g_@@_messages_for_Overleaf_bool

```

```

10596         { \msg_info:nn { nicematrix } { #2~+ } }
10597     }
10598 }
10599 \@@_msg_new:nn { key~without~+~exists }
10600 {
10601     The~key~'\tl_trim_right_spaces:V \l_tmpa_str'~exists~but~does~not~accept~an~
10602     additive~syntax~(with~+=)~.~It~will~be~ignored.
10603 }
10604 \@@_msg_new:nn { key~with~normal~form~exists }
10605 {
10606     No~key~'\l_keys_key_str'.\\
10607     It~will~be~ignored.~Maybe~you~want~to~use~the~key~'\l_tmpa_str'.
10608 }
10609 \str_const:Ne \c_@@_available_keys_str
10610 {
10611     \bool_if:nT { ! \g_@@_messages_for_Overleaf_bool }
10612     { For~a~list~of~the~available~keys,~type~H~<return>. }
10613 }
10614 \seq_new:N \g_@@_types_of_matrix_seq
10615 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
10616 {
10617     NiceMatrix ,
10618     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
10619 }
10620 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
10621 { \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`.

```

10622 \cs_new_protected:Npn \@@_err_too_many_cols:
10623 {
10624     \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
10625     { \@@_fatal:nn { too-many-cols-for-array } }
10626     \int_compare:nNnT \l_@@_last_col_int = { -2 }
10627     { \@@_fatal:n { too-many-cols-for-matrix } }
10628     \int_compare:nNnT \l_@@_last_col_int = { -1 }
10629     { \@@_fatal:n { too-many-cols-for-matrix } }
10630     \bool_if:NF \l_@@_last_col_without_value_bool
10631     { \@@_fatal:n { too-many-cols-for-matrix-with-last-col } }
10632 }

```

The following command must *not* be protected since it's used in an error message.

```

10633 \cs_new:Npn \@@_message_hdotsfor:
10634 {
10635     \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
10636     { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
10637       \token_to_str:N \Hbrace \ is~incorrect. }
10638 }
10639 \cs_new_protected:Npn \@@_Hline_in_cell:
10640 { \@@_error:n { Misuse~of~Hline } }
10641 \@@_msg_new:nn { Misuse~of~Hline }
10642 {
10643     Misuse~of~\token_to_str:N \Hline. \\
10644     Error~in~the~row~ \int_use:N \c@iRow\ of~your~\@@_full_name_env:.~
10645     \token_to_str:N \Hline\ (like~\token_to_str:N \hline)~must~be~used~only~
10646     at~the~beginning~of~a~row.~Your~command~will~be~ignored.
10647 }

```

```

10648 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
10649 {
10650   Incompatible-options.\
10651   You-should-not-use~'hvlines',~'rounded-corners'~and~'corners'~at-the-same-time.~
10652   The-output~will~not~be~reliable.
10653 }
10654 \@@_msg_new:nn { Body-alone }
10655 {
10656   \token_to_str:N \Body\ alone. \
10657   You-have-used~\token_to_str:N \Body\ without~\token_to_str:N \CodeBefore.
10658 }
10659 \@@_msg_new:nn { Misuse-of~CodeBefore }
10660 {
10661   Misuse-of~\token_to_str:N \CodeBefore. \
10662   \token_to_str:N \CodeBefore\ must~be~used~only~at~the~beginning~of~
10663   the~environment.
10664 }
10665 \@@_msg_new:nn { NiceTabularX~probably~required }
10666 {
10667   Incorrect-syntax.\
10668   You-probably-want-to-use~\{NiceTabularX\}~whose-first-argument-is-the~
10669   ~width~that~you~wish~for~your~tabular.~But~you~can~also~use~\{NiceTabular\}~
10670   with~the~key~'width'.
10671 }
10672 \@@_msg_new:nn { cellcolor-in-Block }
10673 {
10674   Misuse-of~\token_to_str:N \cellcolor \
10675   You-can't-use~\token_to_str:N \cellcolor\ in~\token_to_str:N \Block\
10676   \bool_if:NTF \l_@@_amp_in_blocks_bool
10677     { (but~you~could~use~it~in~a~sub~block~since~'&~in~blocks'~is~in~force) }
10678     { (it's~possible~in~a~sub~block~when~'&~in~blocks'~is~in~force) }
10679   .~Here,~you~should~use~the~key~'fill'~of~the~block.~Your~command~will~be~ignored.
10680 }
10681 \@@_msg_new:nn { rowcolor-in-Block }
10682 {
10683   Misuse-of~\token_to_str:N \rowcolor \
10684   You-can't-use~\token_to_str:N \rowcolor\ in~\token_to_str:N \Block.~
10685   However,~it's~possible~to~color~the~block~with~its~key~'fill'.~
10686   Your~command~will~be~ignored.
10687 }
10688 \@@_msg_new:nn { key-color-inside }
10689 {
10690   Deleted-key.\
10691   The-key~'color-inside'~(and~its~alias~'colortbl-like')~has~been~deleted~in
10692   ~'nicematrix'~and~must~not~be~used.
10693 }
10694 \@@_msg_new:nn { Misuse-of~FalseRow }
10695 {
10696   Misuse-of~\token_to_str:N \FalseRow \
10697   Error~in~the~row~ \int_use:N \c@iRow\ of~your~\@@_full_name_env:.~
10698   \token_to_str:N \FalseRow\ must~be~used~only~at~the~beginning~of~a~row.~
10699   Your~command~will~be~ignored.
10700 }
10701 \@@_msg_new:nn { Invalid~argument~for~w }
10702 {
10703   Invalid~argument~for~w.\
10704   You-have-used~the~type~of~alignment~'#1'~for~your~column~'w'~
10705   but~only~'c',~'r',~'l'~and~'s'~are~allowed~in~a~column~'w'.~
10706   If~you~go~on,~'c'~will~be~used.
10707 }

```

```

10708 \@@_msg_new:nn { invalid-weight }
10709 {
10710   Unknown-key.\
10711   The-key~' \l_keys_key_str '~of~your~column~X~is~unknown~and~will~be~ignored.~
10712   The-available-keys~are:~l,~c,~r,~t~(=p),~m,~b,~V~
10713   \IfPackageLoadedTF { varwidth }
10714     { (since~'varwidth'~is~loaded)~}
10715     { (if~you~load~'varwidth')~}
10716     and~real~numbers~for~the~weight~of~the~X~column.
10717 }
10718 \@@_msg_new:nn { last-col-not-used }
10719 {
10720   Column~not~used.\
10721   The-key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
10722   in~your~\@@_full_name_env: .~However,~you~can~go~on.
10723 }
10724 \@@_msg_new:nn { too-many-cols-for-matrix-with-last-col }
10725 {
10726   Too-many-columns.\
10727   In~the~row~ \int_eval:n { \c@iRow },~
10728   you~try~to~use~more~columns~
10729   than~allowed~by~your~ \@@_full_name_env: .
10730   \@@_message_hdotsfor: \
10731   The~maximal~number~of~columns~is~ \int_eval:n { \l_@@_last_col_int - 1 }~
10732   (plus~the~exterior~columns).~
10733   But,~maybe,~you~have~forgotten~a~\token_to_str:N \\.
10734 }
10735 \@@_msg_new:nn { too-many-cols-for-matrix }
10736 {
10737   Too-many-columns.\
10738   In~the~row~ \int_eval:n { \c@iRow },~
10739   you~try~to~use~more~columns~than~allowed~by~your~ \@@_full_name_env: .
10740   \@@_message_hdotsfor: \
10741   Recall~that~the~maximal~number~of~columns~for~a~matrix~
10742   (excepted~the~potential~exterior~columns)~is~fixed~by~the~
10743   LaTeX~counter~'MaxMatrixCols'.~
10744   Its~current~value~is~ \int_use:N \c@MaxMatrixCols \
10745   (use~ \token_to_str:N \setcounter \ to~change~that~value).~
10746   But,~maybe,~you~have~forgotten~a~\token_to_str:N \\.
10747 }
10748 \@@_msg_new:nn { too-many-cols-for-array }
10749 {
10750   Too-many-columns.\
10751   In~the~row~ \int_eval:n { \c@iRow },~
10752   ~you~try~to~use~more~columns~than~allowed~by~your~
10753   \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal~number~of~columns~is~
10754   \int_use:N \g_@@_static_num_of_col_int \
10755   \bool_if:nT
10756     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
10757     { (plus~the~exterior~ones)~}
10758   since~the~preamble~is~' \g_@@_user_preamble_tl '~
10759   But,~maybe,~you~have~forgotten~a~\token_to_str:N \\.
10760 }
10761 \@@_msg_new:nn { columns-not-used }
10762 {
10763   Columns~not~used.\
10764   The~preamble~of~your~ \@@_full_name_env: \ is~
10765   '\exp_not:V \g_@@_user_preamble_tl'.~
10766   It~announces~ \int_use:N \g_@@_static_num_of_col_int \
10767   columns~but~you~only~used~ \int_use:N \c@jCol .~The~columns~you~did~not~
10768   used~won't~be~created.~You~won't~have~similar~warning~till~the~end~of~the~document.

```

```

10769 }
10770 }
10771 \@@_msg_new:nn { Misuse-of-NiceTabularNotes }
10772 {
10773   Misuse-of~\token_to_str:N \NiceTabularNotes \\
10774   \token_to_str:N~\NiceTabularNotes\ should-be-used-only-after-at-tabular~
10775   which-uses~notes/no-print`.~ Your-command-will-be-ignored.
10776 }
10777 \@@_msg_new:nn { empty-preamble }
10778 {
10779   Empty-preamble.\\
10780   The-preamble-of-your~ \@@_full_name_env: \ is-empty.
10781 }
10782 \@@_msg_new:nn { in~first~col }
10783 {
10784   Misuse-of~#1\\
10785   You-can't-use-the-command~#1 in-the-first-column~(number~0)~of-the-array.~
10786   Your-command-will-be-ignored.~You-can-try-to-delete-the-key~'first-col'.
10787 }
10788 \@@_msg_new:nn { in~last~col }
10789 {
10790   Misuse-of~#1\\
10791   You-can't-use-the-command~#1 in-the-last-column~(exterior)~of-the-array.~
10792   Your-command-will-be-ignored.~You-can-try-to-delete-the-key~'last-col'.
10793 }
10794 \@@_msg_new:nn { in~first~row }
10795 {
10796   Misuse-of~#1\\
10797   You-can't-use-the-command~#1 in-the-first-row~(number~0)~of-the-array.~
10798   Your-command-will-be-ignored.~You-can-try-to-delete-the-key~'first-row'.
10799 }
10800 \@@_msg_new:nn { in~last~row }
10801 {
10802   Misuse-of~#1\\
10803   You-can't-use-the-command~#1 in-the-last-row~(exterior)~of-the-array.~
10804   Your-command-will-be-ignored.~You-can-try-to-delete-the-key~'last-row'.
10805 }
10806 \@@_msg_new:nn { TopRule~without~booktabs }
10807 {
10808   Misuse-of~#1\\
10809   You-can't-use-the-command~ #1 because~'booktabs'~is-not-loaded.~
10810   You-should-load~'booktabs'~(before~or~after~'nicematrix').~
10811   Your-command-will-be-ignored.
10812 }
10813 \@@_msg_new:nn{ rotate-in~p~col }
10814 {
10815   \token_to_str:N \rotate\ forbidden.\\
10816   You-should-not-use~\token_to_str:N \rotate\ in-a-column-of~type~'p',~
10817   'b',~'m'\IfPackageLoadedTF { varwidth } { ,~'X'~or~'V' } { ~or~'X' }.~
10818   If-you-go-on,~maybe-you-won't-have-the-expected-output.~You-should~
10819   consider-the-classical-command~\token_to_str:N \rotatebox.
10820 }
10821 \@@_msg_new:nn { TopRule~without~tikz }
10822 {
10823   Misuse-of~#1\\
10824   You-can't-use-the-command~ #1 because~TikZ~is-not-loaded.~
10825   You-should-load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.~
10826   \IfPackageLoadedF { booktabs }
10827     { You-should-also-load~'booktabs'~
10828       with~\token_to_str:N \usepackage \{booktabs\}.~ }

```

```

10829     Your~command~will~be~ignored.
10830 }

10831 \@@_msg_new:nn { caption~outside~float }
10832 {
10833     Key~caption~forbidden.\\
10834     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
10835     environment~(such~as~\{table\}).~Your~key~will~be~ignored.
10836 }

10837 \@@_msg_new:nn { short-caption~without~caption }
10838 {
10839     You~should~not~use~the~key~'short-caption'~without~'caption'.~
10840     However,~your~'short-caption'~will~be~used~as~'caption'.
10841 }

10842 \@@_msg_new:nn { double~closing~delimiter }
10843 {
10844     Double~delimiter.\\
10845     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
10846     delimiter.~This~delimiter~will~be~ignored.~You~can~try~to~use~
10847     \token_to_str:N \SubMatrix\ in~the~\token_to_str:N \CodeAfter.
10848 }

10849 \@@_msg_new:nn { delimiter~after~opening }
10850 {
10851     Double~delimiter.\\
10852     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
10853     delimiter.~That~delimiter~will~be~ignored.
10854 }

10855 \@@_msg_new:nn { bad~option~for~line~style }
10856 {
10857     Bad~line~style.\\
10858     Since~you~haven't~loaded~TikZ,~the~only~value~you~can~give~to~'line~style'~
10859     is~'standard'.~Your~key~will~be~ignored.~You~can~load~TikZ~with~
10860     \token_to_str:N \usepackage \{tikz\},~before~or~after~'nicematrix'.
10861 }

10862 \@@_msg_new:nn { draw~trees~with~no~cell~nodes }
10863 {
10864     Incompatible~keys.\\
10865     You~can't~use~the~key~'draw~trees~in~col'~here~because~the~key~'no~cell~nodes'~
10866     is~in~force~(you~should~deactive~the~key~'no~cell~nodes'~whose~only~goal~
10867     is~to~speed~compilation~up).~If~you~go~on,~that~key~will~be~ignored.
10868 }

10869 \@@_msg_new:nn { corners~with~no~cell~nodes }
10870 {
10871     Incompatible~keys.\\
10872     You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
10873     is~in~force~(you~should~deactive~the~key~'no~cell~nodes'~whose~only~goal~
10874     is~to~speed~compilation~up).\\
10875     If~you~go~on,~that~key~will~be~ignored.
10876 }

10877 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
10878 {
10879     Incompatible~keys.\\
10880     You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
10881     is~in~force~(you~should~deactive~the~key~'no~cell~nodes'~whose~only~goal~
10882     is~to~speed~compilation~up).~If~you~go~on,~those~extra~nodes~won't~be~created.
10883 }

10884 \@@_msg_new:nn { Identical~notes~in~caption }
10885 {
10886     Identical~tabular~notes.\\
10887     You~can't~put~several~notes~with~the~same~content~in~
10888     \token_to_str:N \caption \ (but~it's~possible~in~the~main~tabular).~

```

```

10889     If-you-go-on,~the-output-will-probably-be-erroneous.
10890 }
10891 \@@_msg_new:nn { tabularnote~below~the~tabular }
10892 {
10893     \token_to_str:N \tabularnote \ forbidden\
10894     You-can't-use~ \token_to_str:N \tabularnote \ in~the~caption~
10895     of~your~tabular~because~the~caption~will~be~composed~below~
10896     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
10897     key~'caption-above'~in~ \token_to_str:N \NiceMatrixOptions .~
10898     Your~ \token_to_str:N \tabularnote \ will~be~ignored~and~
10899     no~similar~error~will~raised~in~this~document.
10900 }
10901 \@@_msg_new:nn { Unknown~key~for~rules }
10902 {
10903     Unknown~key.\
10904     There-are~only~three~keys~available~here:~'width',~'color'~and~
10905     'fix-vertex'.~Your~key~' \l_keys_key_str '~will~be~ignored.
10906 }
10907 \@@_msg_new:nn { Unknown~key~for~trees }
10908 {
10909     Unknown~key.\
10910     There-are~only~three~keys~available~here:~width~color~and~
10911     rounded-corners.~Your~key~' \l_keys_key_str '~will~be~ignored.
10912 }
10913 % \end{macrocode}
10914 %
10915 %
10916 % \begin{macrocode}
10917 \@@_msg_new:nn { Unknown~key~for~Hbrace }
10918 {
10919     Unknown~key.\
10920     You-have-used~the~key~' \l_keys_key_str '~but~the~only~
10921     keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
10922     and~ \token_to_str:N \Vbrace \ are:~'brace-shift(+)',~'color',~
10923     'horizontal-label(s)',~'shorten'~'shorten-end'~
10924     and~'shorten-start'.
10925 }
10926 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
10927 {
10928     Unknown~key.\
10929     There~is~only~two~keys~available~here:~
10930     'empty'~and~'not-empty'.~Your~key~' \l_keys_key_str '~will~be~ignored.
10931 }
10932 \@@_msg_new:nn { Unknown~key~for~rotate }
10933 {
10934     Unknown~key.\
10935     The~only~keys~available~here~are~'c'~and~'-90'.~
10936     Your~key~' \l_keys_key_str '~will~be~ignored.
10937 }
10938 \@@_msg_new:nnn { Unknown~key~for~custom-line }
10939 {
10940     Unknown~key.\
10941     The~key~' \l_keys_key_str '~is~unknown~in~a~'custom-line'.~
10942     It~you~go~on,~you~will~probably~have~other~errors. \
10943     \c_@@_available_keys_str
10944 }
10945 {
10946     The~available~keys~are~(in~alphabetic~order):~
10947     ccommand,~
10948     color,~
10949     command,~

```



```

10950     dotted,~
10951     letter,~
10952     multiplicity,~
10953     sep-color,~
10954     tikz,~and~total-width.
10955 }
10956 \@@_msg_new:nnn { Unknown~key~for~default-line }
10957 {
10958     Unknown~key.\\
10959     The~key~' \l_keys_key_str '~is~unknown~in~a~'default-line'.~
10960     It~you~go~on,~you~will~probably~have~other~errors. \\
10961     \c_@@_available_keys_str
10962 }
10963 {
10964     The~available~keys~are~(in~alphabetic~order):~
10965     color,~
10966     dotted,~
10967     multiplicity,~
10968     sep-color,~
10969     tikz~(when~TikZ~is~loaded)~
10970     and~total-width.
10971 }
10972 \@@_msg_new:nnn { Unknown~key~for~xdots }
10973 {
10974     Unknown~key.\\
10975     The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
10976     \c_@@_available_keys_str
10977 }
10978 {
10979     The~available~keys~are~(in~alphabetic~order):~
10980     'color',~
10981     'horizontal(s)-labels',~
10982     'inter',~
10983     'line-style',~
10984     'nullify',~
10985     'radius',~
10986     'shorten',~
10987     'shorten-end'~and~'shorten-start'.
10988 }
10989 \@@_msg_new:nn { Unknown~key~for~rowcolors }
10990 {
10991     Unknown~key.\\
10992     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
10993     (and~you~try~to~use~' \l_keys_key_str ').~Your~key~will~be~ignored.
10994 }
10995 \@@_msg_new:nn { Col~outside~tabular~in~trees }
10996 {
10997     Error~with~'draw-trees-in-col' \\
10998     The~number~of~column~'#1'~is~outside~your~tabular~since~the~last~column~
10999     is~\int_use:N \c@jCol.~It~will~be~ignored.
11000 }
11001 \@@_msg_new:nn { Last~col~in~trees }
11002 {
11003     Error~with~'draw-trees-in-col' \\
11004     You~can't~use~'draw-trees-in-col'~with~the~column~'#1'~ because~it's~
11005     the~last~column~of~your~tabular.~It~will~be~ignored.
11006 }
11007 \@@_msg_new:nn { label~without~caption }
11008 {
11009     You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
11010     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
11011 }

```

```

11012 \@@_msg_new:nn { W-warning }
11013 {
11014   Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
11015   (row~ \int_use:N \c@iRow ).
11016 }
11017 \@@_msg_new:nn { Construct-too-large }
11018 {
11019   Construct-too-large.\\
11020   Your~command~ \token_to_str:N #1
11021   can't~be~drawn~because~your~matrix~is~too~small.~
11022   Your~command~will~be~ignored.
11023 }
11024 \@@_msg_new:nn { underscore-after-nicematrix }
11025 {
11026   Problem-with-'underscore'.\\
11027   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
11028   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
11029   ' \token_to_str:N \Cdots \token_to_str:N _
11030   \{ n \token_to_str:N \text \{ ~times \} \}' .
11031 }
11032 \@@_msg_new:nn { ampersand-in-light-syntax }
11033 {
11034   Ampersand-forbidden.\\
11035   You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
11036   ~the~key~'light-syntax'~is~in~force.
11037 }
11038 \@@_msg_new:nn { double-backslash-in-light-syntax }
11039 {
11040   Double-backslash-forbidden.\\
11041   You~can't~use~ \token_to_str:N \\
11042   ~to~separate~rows~because~the~key~'light-syntax'~
11043   is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
11044   (set~by~the~key~'end-of-row').
11045 }
11046 \@@_msg_new:nn { bad-value-for-baseline }
11047 {
11048   Bad-value-for-baseline.\\
11049   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
11050   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
11051   \int_use:N \g_@@_row_total_int \ or~equal~to~'t',~'c'~or~'b'~or~of~
11052   the~form~'line-i'.~A~value~of~1~will~be~used.
11053 }
11054 \@@_msg_new:nn { bad-value-for-baseline-line }
11055 {
11056   Bad-value-for-baseline-with-line.\\
11057   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
11058   valid.~The~number~of~the~line~must~be~between~1~and~
11059   \int_eval:n { \c@iRow + 1 }.~
11060   A~value~of~'line-1'~will~be~used.
11061 }
11062 \@@_msg_new:nn { detection-of-empty-cells }
11063 {
11064   Problem-with-'not-empty'\\
11065   For~technical~reasons,~you~must~activate~
11066   'create-cell-nodes'~in~ \token_to_str:N \CodeBefore \
11067   in~order~to~use~the~key~' \l_keys_key_str '~
11068   Your~key~will~be~ignored.
11069 }
11070 \@@_msg_new:nn { siunitx-too-old }
11071 {

```

```

11072     siunitx-too-old\\
11073     You~can't~use~the~columns~'S'~because~your~version~of~'siunitx'~
11074     is~too~old.~You~need~at~least~the~version~3.5.1~(2026/03/26).
11075 }
11076 \@@_msg_new:nn { Invalid~name }
11077 {
11078     Invalid~name.\\
11079     You~can't~give~the~name~' \l_keys_value_tl '~to~a~ \token_to_str:N
11080     \SubMatrix \ of~your~ \@@_full_name_env: .~
11081     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
11082     Your~key~will~be~ignored.
11083 }
11084 \@@_msg_new:nn { Hbrace~not~allowed }
11085 {
11086     Command~not~allowed.\\
11087     You~can't~use~the~command~ \token_to_str:N #1
11088     because~you~have~not~loaded~
11089     \IfPackageLoadedTF { tikz }
11090     { the~TikZ~library~'decorations.pathreplacing'~Use~ }
11091     { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
11092     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\
11093     Your~command~will~be~ignored.
11094 }
11095 \@@_msg_new:nn { Vbrace~not~allowed }
11096 {
11097     Command~not~allowed.\\
11098     You~can't~use~the~command~ \token_to_str:N \Vbrace \
11099     because~you~have~not~loaded~TikZ~
11100     and~the~TikZ~library~'decorations.pathreplacing'~
11101     Use: ~\token_to_str:N \usepackage \{tikz\}~
11102     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
11103     Your~command~will~be~ignored.
11104 }
11105 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
11106 {
11107     Wrong~line.\\
11108     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
11109     \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
11110     number~is~not~valid.~It~will~be~ignored.
11111 }
11112 \@@_msg_new:nn { Impossible~SubMatrix }
11113 {
11114     Impossible~SubMatrix.\\
11115     It's~impossible~to~draw~your~\token_to_str:N \SubMatrix \
11116     because~all~the~cells~are~empty~in~the~column~on~the~#1~
11117     side~of~that~\token_to_str:N \SubMatrix.
11118     \bool_if:NT \l_@@_submatrix_slim_bool
11119     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
11120     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
11121 }
11122 \@@_msg_new:nn { Impossible~SubMatrix~no~cell~nodes }
11123 {
11124     Impossible~SubMatrix.\\
11125     It's~impossible~to~draw~your~ \token_to_str:N \SubMatrix \
11126     because~the~key~'no~cell~nodes'~is~in~force.~
11127     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
11128 }
11129 \@@_msg_new:nnn { width~without~X~columns }
11130 {
11131     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
11132     the~preamble~(' \exp_not:V \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .~

```

```

11133     Your~key~will~be~ignored.
11134 }
11135 {
11136     This~message~is~the~message~'width~without~X~columns'~
11137     of~the~module~'nicematrix'.~
11138     The~experimented~users~can~disable~that~message~with~
11139     \token_to_str:N \msg_redirect_name:nnn .\\
11140 }
11141
11142 \@@_msg_new:nn { key~multiplicity~with~dotted }
11143 {
11144     Incompatible~keys. \\
11145     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
11146     in~a~'custom~line'.~They~are~incompatible.~
11147     The~key~'multiplicity'~will~be~ignored.
11148 }
11149 \@@_msg_new:nn { empty~environment }
11150 {
11151     Empty~environment.\\
11152     Your~ \@@_full_name_env: \ is~empty.
11153 }
11154 \@@_msg_new:nn { No~letter~and~no~command }
11155 {
11156     Erroneous~use.\\
11157     Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
11158     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
11159     '~ccommand'~(to~draw~horizontal~rules).~However,~you~can~go~on.
11160 }
11161 \@@_msg_new:nn { Forbidden~letter }
11162 {
11163     Forbidden~letter.\\
11164     You~can't~use~the~letter~'#1'~for~a~customized~line.~
11165     It~will~be~ignored.~The~forbidden~letters~are:~\c_@@_forbidden_letters_str
11166 }
11167 \@@_msg_new:nn { Several~letters }
11168 {
11169     Wrong~name.\\
11170     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
11171     have~used~' \l_@@_letter_str ').~It~will~be~ignored.
11172 }
11173 \@@_msg_new:nn { Delimiter~with~small }
11174 {
11175     Delimiter~forbidden.\\
11176     You~can't~put~a~delimiter~in~the~preamble~of~your~
11177     \@@_full_name_env: \
11178     because~the~key~'small'~is~in~force.
11179 }
11180 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
11181 {
11182     Unknown~cell.\\
11183     Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
11184     the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
11185     can't~be~executed~because~a~cell~doesn't~exist.~
11186     Your~command~ \token_to_str:N \line \ will~be~ignored.
11187 }
11188 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
11189 {
11190     Duplicate~name. \\
11191     The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
11192     in~this~ \@@_full_name_env: .~Your~key~will~be~ignored.~

```

```

11193 \bool_if:NF \g_@@_messages_for_Overleaf_bool
11194 { For~a~list~of~the~names~already~used,~type~H~<return>. }
11195 }
11196 {
11197   The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
11198   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
11199 }
11200 \@@_msg_new:nn { r~or~l~with~preamble }
11201 {
11202   Erroneous~use. \\
11203   You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
11204   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
11205   your~ \@@_full_name_env: .~
11206   Your~key~will~be~ignored.
11207 }
11208 \@@_msg_new:nn { Hdotsfor~in~col~0 }
11209 {
11210   Erroneous~use. \\
11211   You~can't~use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\
11212   in~an~exterior~column~of~the~array.
11213 }
11214 \@@_msg_new:nn { bad~corner }
11215 {
11216   Bad~corner. \\
11217   '#1'~is~an~incorrect~specification~for~a~corner~(in~the~key~
11218   'corners').~The~available~values~are:~NW,~SW,~NE,~SE,~N,~S,~W~and~E~
11219   This~specification~of~corner~will~be~ignored.
11220 }
11221 \@@_msg_new:nn { bad~border }
11222 {
11223   Bad~border.\\
11224   '\l_keys_key_str'~is~an~incorrect~specification~for~a~border~
11225   (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
11226   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
11227   also~use~the~key~'tikz'
11228   \IfPackageLoadedF { tikz }
11229   { ~if~you~load~the~LaTeX~package~'tikz' } ).~
11230   This~specification~of~border~will~be~ignored.
11231 }
11232 \@@_msg_new:nn { TikzEveryCell~without~tikz }
11233 {
11234   TikZ~not~loaded. \\
11235   You~can't~use~ \token_to_str:N \TikzEveryCell \
11236   because~you~have~not~loaded~TikZ.~You~can~load~TikZ~with~
11237   \token_to_str:N \usepackage \{tikz\},~before~or~after~'nicematrix'.~
11238   Your~command~will~be~ignored.
11239 }
11240 \@@_msg_new:nn { tikz~key~without~tikz }
11241 {
11242   TikZ~not~loaded. \\
11243   You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N
11244   \Block '~because~you~have~not~loaded~TikZ.~
11245   You~can~load~TikZ~with~\token_to_str:N \usepackage \{tikz\},~
11246   before~or~after~'nicematrix'.~Your~key~will~be~ignored.
11247 }
11248 \@@_msg_new:nn { Bad~argument~for~Block }
11249 {
11250   Bad~argument. \\
11251   The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
11252   be~of~the~form~'i-j'~(or~totally~empty)~and~you~have~used:~
11253   '#1'.~ If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono-cell~(as~if~

```

```

11254     the~argument~was~empty).
11255 }
11256 \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
11257 {
11258     Erroneous~use. \\\
11259     In~the~ \@@_full_name_env: ,~you~must~use~the~key~
11260     'last-col'~without~value.~However,~you~can~go~on~for~this~time~
11261     (the~value~' \l_keys_value_tl '~will~be~ignored).
11262 }
11263 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
11264 {
11265     Erroneous~use. \\\
11266     In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
11267     'last-col'~without~value.~ However,~you~can~go~on~for~this~time~
11268     (the~value~' \l_keys_value_tl '~will~be~ignored).
11269 }
11270 \@@_msg_new:nn { Block-too-large-1 }
11271 {
11272     Block~too~large. \\\
11273     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
11274     too~small~for~that~block.~This~block~and~maybe~others~will~be~ignored.
11275 }
11276 \@@_msg_new:nn { Block-too-large-2 }
11277 {
11278     Block~too~large. \\\
11279     The~preamble~of~your~ \@@_full_name_env: \ announces~ \int_use:N
11280     \g_@@_static_num_of_col_int \
11281     columns~but~you~use~only~ \int_use:N \c@jCol \ and~that's~why~a~block~
11282     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
11283     (&)~at~the~end~of~the~first~row~of~your~ \@@_full_name_env: .~
11284     This~block~and~maybe~others~will~be~ignored.
11285 }
11286 \@@_msg_new:nn { unknown~column~type }
11287 {
11288     Bad~column~type. \\\
11289     The~column~type~'#1'~in~your~ \@@_full_name_env: \ is~unknown.
11290 }
11291 \@@_msg_new:nn { unknown~column~type~multicolumn }
11292 {
11293     Bad~column~type. \\\
11294     The~column~type~'#1'~in~the~command~\token_to_str:N \multicolumn \
11295     ~of~your~ \@@_full_name_env: \ is~unknown.
11296 }
11297 \@@_msg_new:nn { unknown~column~type~S }
11298 {
11299     Bad~column~type. \\\
11300     The~column~type~'S'~in~your~ \@@_full_name_env: \ is~unknown.~
11301     If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
11302     load~that~package~with~\token_to_str:N \usepackage \{siunitx\}.
11303 }
11304 \@@_msg_new:nn { unknown~column~type~S~multicolumn }
11305 {
11306     Bad~column~type. \\\
11307     The~column~type~'S'~in~the~command~\token_to_str:N \multicolumn \
11308     of~your~ \@@_full_name_env: \ is~unknown.~If~you~want~to~use~the~column~
11309     type~'S'~of~siunitx,~you~should~load~that~package~with~
11310     \token_to_str:N \usepackage \{siunitx\}.
11311 }
11312 \@@_msg_new:nn { tabularnote~forbidden }
11313 {

```

```

11314   Forbidden-command. \\
11315   You-can't-use-the-command~ \token_to_str:N \tabularnote \
11316   ~here.~This-command-is-available-only-in~
11317   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
11318   the-argument-of-a-command~\token_to_str:N \caption \ included-
11319   in-an-environment~\{table\}.~Your-command-will-be-ignored.
11320 }
11321 \@@_msg_new:nn { borders~forbidden }
11322 {
11323   Forbidden-key.\\
11324   You-can't-use-the-key~'borders'~of-the-command~ \token_to_str:N \Block \
11325   because-the-option~'rounded-corners'~is~in~force~with~a~non-zero~value.~
11326   Your-key-will-be-ignored.
11327 }
11328 \@@_msg_new:nn { bottomrule~without~booktabs }
11329 {
11330   booktabs-not-loaded.\\
11331   You-can't-use-the-key~'tabular/bottomrule'~because-you-haven't~
11332   loaded~'booktabs'.~You-should-load~'booktabs',~before~or~
11333   after~'nicematrix'.~Your-key-will-be-ignored.
11334 }
11335 \@@_msg_new:nn { enumitem~not~loaded }
11336 {
11337   enumitem-not-loaded. \\
11338   You-can't-use-the-command~ \token_to_str:N \tabularnote \
11339   ~because-you-haven't~loaded~'enumitem'.~We-should-load-it~
11340   (before~or~after~'nicematrix').~All-the-commands~
11341   \token_to_str:N \tabularnote \ will-be-ignored-in-the-document.
11342 }
11343 \@@_msg_new:nn { tikz~without~tikz }
11344 {
11345   TikZ-not-loaded. \\
11346   You-can't-use-the-key~'tikz'~here~because~TikZ-is~not~loaded.~You~
11347   should~load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.~
11348   If-you-go-on,~your-key-will-be-ignored.
11349 }
11350 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
11351 {
11352   TikZ-not-loaded. \\
11353   You-have-used-the-key~'tikz'~in-the-definition-of~a~
11354   customized~line~(with~'custom-line')~but~TikZ-is~not~loaded.~
11355   You-can-go-on~but~you-will~have~another~error~if~you~actually~
11356   use~that~custom~line.~You-should-load~TikZ~with~
11357   \token_to_str:N \usepackage \{tikz\}.
11358 }
11359 \@@_msg_new:nn { tikz~in~borders~without~tikz }
11360 {
11361   TikZ-not-loaded. \\
11362   You-have-used-the-key~'tikz'~in~a~key~'borders'~(of~a~
11363   command~' \token_to_str:N \Block ')~but~TikZ-is~not~loaded.~
11364   Your-key-will-be-ignored.~You-should-load~TikZ~with~
11365   \token_to_str:N \usepackage \{tikz\}
11366 }
11367 \@@_msg_new:nn { color~in~custom~line~with~tikz }
11368 {
11369   Erroneous-use.\\
11370   In~a~'custom-line',~you-have-used~both~'tikz'~(or~'dashed')~and~'color',~
11371   which-is-forbidden~(you-should-use~'color'~inside~the~key~'tikz'~itself).~
11372   The-key~'color'~will-be-ignored.
11373 }

```

```

11374 \@@_msg_new:nn { Wrong-last-row }
11375 {
11376   Wrong-number.\
11377   You-have-used-'last-row'= \int_use:N \l_@@_last_row_int '~but-your~
11378   \@@_full_name_env: \ seems-to-have~ \int_use:N \c@iRow \ rows.~
11379   If-you-go-on,~the-value-of~ \int_use:N \c@iRow \ will-be-used-for~
11380   last-row-but-you-should-correct-your-code.~You-can-avoid-this~
11381   problem-by-using-'last-row'~without-value~(more-compilations~
11382   might-be-necessary).
11383 }
11384 \@@_msg_new:nn { Yet-in-env }
11385 {
11386   Nested-environments.\
11387   Environments-of~nicematrix~can't-be-nested.~However-you~
11388   can-insert,~for-example,~a~\{tabular\}~in-a~\{NiceTabular\}~
11389   or~a~\{NiceTabular\}~in-a~\{tabular\}.~You-can-also-compose~
11390   an-environment-of~nicematrix~in-a-box-of~LaTeX~and-insert~
11391   that~box~in~another-environment-of~nicematrix.
11392 }
11393 \@@_msg_new:nn { Outside-math-mode }
11394 {
11395   Outside-math-mode.\
11396   The~\@@_full_name_env: \ can-be-used-only-in-math-mode~
11397   (and-not-in~ \token_to_str:N \vcenter ).
11398 }
11399 \@@_msg_new:nn { One-letter-allowed }
11400 {
11401   Bad-name.\
11402   The-value-of-key~' \l_keys_key_str '~must-be-of-length-1~and~
11403   you-have-used~' \l_keys_value_tl '~.~Your-key-will-be-ignored.
11404 }
11405 \@@_msg_new:nn { TabularNote-in-CodeAfter }
11406 {
11407   Environment~\{TabularNote\}~forbidden.\
11408   You-must-use~\{TabularNote\}~at-the-end-of-your~\{NiceTabular\}~
11409   but~*before*~the~ \token_to_str:N \CodeAfter .~Your-environment~
11410   \{TabularNote\}~will-be-ignored.
11411 }
11412 \@@_msg_new:nn { varwidth-not-loaded }
11413 {
11414   varwidth-not-loaded.\
11415   You-can't-use-the-column-type~'V'~because~'varwidth'~is-not~
11416   loaded.~You-should-load~'varwidth',~before-or-after~'nicematrix'.~
11417   Your-column-will-behave-like~'p'.
11418 }
11419 \@@_msg_new:nn { varwidth-not-loaded-in-X }
11420 {
11421   varwidth-not-loaded.\
11422   You-can't-use-the-key~'V'~in-your-column~'X'~
11423   because~'varwidth'~is-not-loaded.~You-should-load~'varwidth',~
11424   before-or-after~'nicematrix'.~ Your-key-will-be-ignored.
11425 }
11426 \@@_msg_new:nnn { Unknown-key-for-a-rule }
11427 {
11428   Unknown-key.\
11429   Your-key~' \l_keys_key_str '~is-unknown-for-a-rule.\
11430   \c_@@_available_keys_str
11431 }
11432 {
11433   The-available-keys-are:~color,~multiplicity,~sep-color,~tikz~
11434   and-total-width~(the-latter-is-meaningful-only-in-cunjunction-with-'tikz').

```


11435 }

If fact, there is also the key dotted but it won't be very useful since we provide `\hdottedline`, `\cdottedline` and the letter `..`.

```

11436 \@@_msg_new:nnn { Unknown~key~for~Block }
11437 {
11438   Unknown~key. \\
11439   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11440   \token_to_str:N \Block .~Your~key~will~be~ignored. \\
11441   \c_@@_available_keys_str
11442 }
11443 {
11444   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
11445   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~name,~
11446   opacity,~rounded~corners,~r,~respect~arraystretch,~rules/width,~t,~T,~tikz,~
11447   transparent~and~vlines.
11448 }
11449 \@@_msg_new:nnn { Unknown~key~for~Brace }
11450 {
11451   Unknown~key. \\
11452   The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
11453   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace .~
11454   Your~key~will~be~ignored. \\
11455   \c_@@_available_keys_str
11456 }
11457 {
11458   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
11459   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
11460   right~shorten)~and~yshift.
11461 }
11462 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
11463 {
11464   Unknown~key. \\
11465   The~key~' \l_keys_key_str '~is~unknown.~It~will~be~ignored. \\
11466   \c_@@_available_keys_str
11467 }
11468 {
11469   The~available~keys~are~(in~alphabetic~order):~
11470   delimiters/color,~
11471   rules~(with~the~subkeys~'color'~and~'width'),~
11472   sub~matrix~(several~subkeys)~
11473   and~xdots~(several~subkeys).~
11474   The~latter~is~for~the~command~ \token_to_str:N \line .
11475 }
11476 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
11477 {
11478   Unknown~key. \\
11479   The~key~' \l_keys_key_str '~is~unknown.~Your~key~will~be~ignored. \\
11480   \c_@@_available_keys_str
11481 }
11482 {
11483   The~available~keys~are~(in~alphabetic~order):~
11484   create~cell~nodes,~
11485   delimiters/color~and~
11486   sub~matrix~(several~subkeys).
11487 }
11488 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
11489 {
11490   Unknown~key. \\
11491   The~key~' \l_keys_key_str '~is~unknown.~Your~key~will~be~ignored. \\
11492   \c_@@_available_keys_str

```

```

11493 }
11494 {
11495     The-available-keys-are~(in~alphabetic-order):~
11496     'delimiters/color',~
11497     'extra-height',~
11498     'hlines',~
11499     'hvlines',~
11500     'left-xshift',~
11501     'name',~
11502     'right-xshift',~
11503     'rules'~(with~the~subkeys~'color'~and~'width'),~
11504     'slim',~
11505     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~and~'right-xshift').
11506 }
11507 \@@_msg_new:nnn { Unknown~key~for~notes }
11508 {
11509     Unknown~key.\\
11510     The~key~' \l_keys_key_str '~is~unknown.~ Your~key~will~be~ignored. \\
11511     \c_@@_available_keys_str
11512 }
11513 {
11514     The-available-keys-are~(in~alphabetic-order):~
11515     bottomrule,~
11516     code-after,~
11517     code-before(+),~
11518     detect-duplicates,~
11519     enumitem-keys,~
11520     enumitem-keys-para,~
11521     para,~
11522     label-in-list,~
11523     label-in-tabular~and~
11524     style.
11525 }
11526 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
11527 {
11528     Unknown~key.\\
11529     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11530     \token_to_str:N \RowStyle .~Your~key~will~be~ignored. \\
11531     \c_@@_available_keys_str
11532 }
11533 {
11534     The-available-keys-are~(in~alphabetic-order):~
11535     bold,~
11536     cell-space-top-limit(+),~
11537     cell-space-bottom-limit(+),~
11538     cell-space-limits(+),~
11539     color,~
11540     fill~(alias:~rowcolor),~
11541     nb-rows,~
11542     opacity~and~
11543     rounded-corners.
11544 }
11545 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
11546 {
11547     Unknown~key.\\
11548     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11549     \token_to_str:N \NiceMatrixOptions .~Your~key~will~be~ignored. \\
11550     \c_@@_available_keys_str
11551 }
11552 {
11553     The-available-keys-are~(in~alphabetic-order):~
11554     &-in-blocks,~
11555     allow-duplicate-names,~

```

```

11556    ampersand-in-blocks,~
11557    caption-above,~
11558    cell-space-bottom-limit(+),~
11559    cell-space-limits(+),~
11560    cell-space-top-limit(+),~
11561    code-for-first-col(+),~
11562    code-for-first-row(+),~
11563    code-for-last-col(+),~
11564    code-for-last-row(+),~
11565    corners,~
11566    custom-key,~
11567    create-extra-nodes,~
11568    create-medium-nodes,~
11569    create-large-nodes,~
11570    custom-line,~
11571    delimiters~(several~subkeys),~
11572    end-of-row,~
11573    first-col,~
11574    first-row,~
11575    h(v)lines,~
11576    h(v)lines-except-borders,~
11577    last-col,~
11578    last-row,~
11579    left-margin,~
11580    light-syntax,~
11581    light-syntax-expanded,~
11582    matrix/columns-type,~
11583    no-cell-nodes,~
11584    notes~(several~subkeys),~
11585    nullify-dots,~
11586    pgf-node-code,~
11587    renew-dots,~
11588    renew-matrix,~
11589    respect-arraystretch,~
11590    rounded-corners,~
11591    right-margin,~
11592    rules~(with~the~subkeys~'color'~and~'width'),~
11593    small,~
11594    sub-matrix~(several~subkeys),~
11595    vlines,~
11596    width-of-false,~
11597    xdots~(several~subkeys).
11598 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no `l` and `r`.

```

11599 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
11600 {
11601   Unknown~key.\\
11602   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11603   \{NiceArray\}.~Your~key~will~be~ignored. \\
11604   \c_@@_available_keys_str
11605 }
11606 {
11607   The~available~keys~are~(in~alphabetic~order):~
11608   &-in-blocks,~
11609   ampersand-in-blocks,~
11610   b,~
11611   baseline,~
11612   c,~
11613   cell-space-bottom-limit,~
11614   cell-space-limits,~
11615   cell-space-top-limit,~
11616   code-after,~

```

```

11617 code-for-first-col(+),~
11618 code-for-first-row(+),~
11619 code-for-last-col(+),~
11620 code-for-last-row(+),~
11621 columns-width,~
11622 corners,~
11623 create-blocks-in-col,~
11624 create-extra-nodes,~
11625 create-medium-nodes,~
11626 create-large-nodes,~
11627 draw-trees-in-col,~
11628 extra-left-margin,~
11629 extra-right-margin,~
11630 first-col,~
11631 first-row,~
11632 h(v)lines,~
11633 h(v)lines-except-borders,~
11634 last-col,~
11635 last-row,~
11636 left-margin,~
11637 light-syntax,~
11638 light-syntax-expanded,~
11639 name,~
11640 no-cell-nodes,~
11641 nullify-dots,~
11642 pgf-node-code,~
11643 renew-dots,~
11644 respect-arraystretch,~
11645 right-margin,~
11646 rounded-corners,~
11647 rules~(with~the~subkeys~'color'~and~'width'),~
11648 small,~
11649 t,~
11650 vlines,~
11651 width-of-false,~
11652 xdots/color,~
11653 xdots/shorten-start(+),~
11654 xdots/shorten-end(+),~
11655 xdots/shorten(+)-and~
11656 xdots/line-style.
11657 }

```

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`).

```

11658 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
11659 {
11660   Unknown~key.\\
11661   The~key~' \l_keys_key_str '~is~unknown~for~the~
11662   \@@_full_name_env: .~Your~key~will~be~ignored. \\
11663   \c_@@_available_keys_str
11664 }
11665 {
11666   The~available~keys~are~(in~alphabetic~order):~
11667   &-in-blocks,~
11668   ampersand-in-blocks,~
11669   b,~
11670   baseline,~
11671   c,~
11672   cell-space-bottom-limit,~
11673   cell-space-limits,~
11674   cell-space-top-limit,~
11675   code-after,~
11676   code-for-first-col(+),~

```

```

11677 code-for-first-row(+),~
11678 code-for-last-col(+),~
11679 code-for-last-row(+),~
11680 columns-type,~
11681 columns-width,~
11682 corners,~
11683 create-blocks-in-col,~
11684 create-extra-nodes,~
11685 create-medium-nodes,~
11686 create-large-nodes,~
11687 draw-trees-in-col,~
11688 extra-left-margin,~
11689 extra-right-margin,~
11690 first-col,~
11691 first-row,~
11692 h(v)lines,~
11693 h(v)lines-except-borders,~
11694 l,~
11695 last-col,~
11696 last-row,~
11697 left-margin,~
11698 light-syntax,~
11699 light-syntax-expanded,~
11700 name,~
11701 no-cell-nodes,~
11702 nullify-dots,~
11703 pgf-node-code,~
11704 r,~
11705 renew-dots,~
11706 respect-arraystretch,~
11707 right-margin,~
11708 rounded-corners,~
11709 rules~(with~the~subkeys~'color'~and~'width'),~
11710 small,~
11711 t,~
11712 vlines,~
11713 width-of-false,~
11714 xdots/color,~
11715 xdots/shorten-start(+),~
11716 xdots/shorten-end(+),~
11717 xdots/shorten(+)-and~
11718 xdots/line-style.
11719 }

11720 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
11721 {
11722   Unknown~key.\\
11723   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11724   \{NiceTabular\}.~Your~key~will~be~ignored. \\
11725   \c_@@_available_keys_str
11726 }
11727 {
11728   The~available~keys~are~(in~alphabetic~order):~
11729   &~in~blocks,~
11730   ampersand~in~blocks,~
11731   b,~
11732   baseline,~
11733   c,~
11734   caption,~
11735   cell-space-bottom-limit,~
11736   cell-space-limits,~
11737   cell-space-top-limit,~
11738   code~after,~
11739   code~for~first~col(+),~

```

```

11740 code-for-first-row(+),~
11741 code-for-last-col(+),~
11742 code-for-last-row(+),~
11743 columns-width,~
11744 corners,~
11745 custom-line,~
11746 create-blocks-in-col,~
11747 create-extra-nodes,~
11748 create-medium-nodes,~
11749 create-large-nodes,~
11750 draw-trees-in-col,~
11751 extra-left-margin,~
11752 extra-right-margin,~
11753 first-col,~
11754 first-row,~
11755 h(v)lines,~
11756 h(v)lines-except-borders,~
11757 label,~
11758 last-col,~
11759 last-row,~
11760 left-margin,~
11761 light-syntax,~
11762 light-syntax-expanded,~
11763 name,~
11764 no-cell-nodes,~
11765 notes~(several~subkeys),~
11766 nullify-dots,~
11767 pgf-node-code,~
11768 renew-dots,~
11769 respect-arraystretch,~
11770 right-margin,~
11771 rounded-corners,~
11772 rules~(with~the~subkeys~'color'~and~'width'),~
11773 short-caption,~
11774 t,~
11775 tabularnote,~
11776 vlines,~
11777 width-of-false,~
11778 xdots/color,~
11779 xdots/shorten-start(+),~
11780 xdots/shorten-end(+),~
11781 xdots/shorten(+)-and~
11782 xdots/line-style.
11783 }

11784 \@@_msg_new:nnn { Duplicate-name }
11785 {
11786   Duplicate-name.\
11787   The-name~' \l_keys_value_tl 'is-already-used-and-you-shouldn't-use~
11788   the-same-environment-name-twice.~You-can-go-on,~but,~
11789   maybe,~you-will-have-incorrect-results-especially~
11790   if-you-use-'columns-width=auto'.~If-you-don't-want-to-see-this~
11791   message-again,~use-the-key~'allow-duplicate-names'~in~
11792   ' \token_to_str:N \NiceMatrixOptions '.\
11793   \bool_if:NF \g_@@_messages_for_Overleaf_bool
11794     { For-a-list-of-the-names-already-used,~type-H~<return>. }
11795 }
11796 {
11797   The-names-already-defined-in-this-document-are:~
11798   \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
11799 }

11800 \@@_msg_new:nn { caption-above~in~env }
11801 {
11802   The-key~'caption-above'~must-be-used-in~\token_to_str:N \NiceMatrixOptions.~

```

```

11803     Your~key~will~be~ignored.
11804 }
11805 \@@_msg_new:nn { show-cell-names }
11806 {
11807     There~is~no~key~'show-cell-names'~in~nicematrix.\\
11808     You~should~use~the~command~\token_to_str:N \ShowCellNames\
11809     in~the~\token_to_str:N \CodeBefore\ or~the~\token_to_str:N
11810     \CodeAfter.~Your~key~will~be~ignored.
11811 }
11812 \@@_msg_new:nn { Option~auto~for~columns~width }
11813 {
11814     Erroneous~use.\\
11815     You~can't~give~the~value~'auto'~to~the~key~'columns~width'~here.~
11816     Your~key~will~be~ignored.
11817 }
11818 \@@_msg_new:nn { NiceTabularX~without~X }
11819 {
11820     NiceTabularX~without~X.\\
11821     You~should~not~use~\{NiceTabularX\}~without~X~columns.~However,~you~can~go~on.
11822 }
11823 \@@_msg_new:nn { Preamble~forgotten }
11824 {
11825     Preamble~forgotten.\\
11826     You~have~probably~forgotten~the~preamble~of~your~
11827     \@@_full_name_env: .
11828 }
11829 \@@_msg_new:nn { Invalid~col~number }
11830 {
11831     Invalid~column~number.\\
11832     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11833     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.~
11834     Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
11835 }
11836 \@@_msg_new:nn { Invalid~row~number }
11837 {
11838     Invalid~row~number.\\
11839     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11840     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.~
11841     Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
11842 }
11843 \@@_define_com:NNN p ( )
11844 \@@_define_com:NNN b [ ]
11845 \@@_define_com:NNN v | |
11846 \@@_define_com:NNN V \l \l
11847 \@@_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>	38
9	The <code>\CodeBefore</code>	54
10	The environment <code>{NiceArrayWithDelims}</code>	58
11	Construction of the preamble of the array	63
12	The redefinition of <code>\multicolumn</code>	81
13	The environment <code>{NiceMatrix}</code> and its variants	98
	13.1 Definition of <code>{pNiceMatrix}</code>	98
	13.2 The key <code>renew-matrix</code>	99
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	99
15	After the construction of the array	100
16	We draw the dotted lines	107
17	The actual instructions for drawing the dotted lines with <code>TikZ</code>	125
18	User commands available in the new environments	130
19	The command <code>\line</code> accessible in <code>\CodeAfter</code>	137
20	The command <code>\RowStyle</code>	138
21	Colors of cells, rows and columns	141
22	The vertical and horizontal rules	154
23	The empty corners	176
24	The environment <code>{NiceMatrixBlock}</code>	179
25	The extra nodes	180
26	The blocks	185
27	Automatic arrays	213
28	The redefinition of the command <code>\dotfill</code>	214
29	The command <code>\diagbox</code>	215

30	The keyword <code>\CodeAfter</code>	216
31	The delimiters in the preamble	217
32	The command <code>\SubMatrix</code>	218
33	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	227
34	The commands <code>HBrace</code> et <code>VBrace</code>	230
35	The command <code>TikzEveryCell</code>	233
36	The key <code>draw-trees-in-col</code>	235
37	The key <code>create-blocks-in-col</code>	236
38	The command <code>\ShowCellNames</code>	237
39	We process the options at package loading	239
40	About the package underscore	240
41	Compatibility with <code>threeparttable</code>	241
42	Gestion of the errors	241
	42.1 Security in the <code>CodeBefore</code> and the <code>CodeAfter</code>	241
	42.2 The error messages of the package	242