

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

September 21, 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~2026-06-01. \\
12   If~you~use~Overleaf,~you~need~at~least~"TeXLive~2026".\\
13   The~package~'nicematrix'~won't~be~loaded.
14 }
15 \IfFormatAtLeastTF { 2026-06-01 }
16 { }
17 { \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.

```
18 \RequirePackage { amsmath }
```

*This document corresponds to the version 7.11d of `nicematrix`, at the date of 2026/09/21.

```

19 \msg_new:nnn { nicematrix } { array-too-old }
20 {
21   Your~version-of~'array'~is~too~old. \
22   You~need~at~least~the~version~of~2025-09-25~and~the~version~that~
23   you~have~loaded~has~the~date:~
24   \str_range:cnn { ver @ array . sty } { 1 } { 10 }.~
25   The~package~'nicematrix'~won't~be~loaded.
26 }
27 \RequirePackage{array}
28 \IfPackageAtLeastF { array }
29 { 2025/09/25 }
30 { \msg_critical:nn { nicematrix } { array-too-old } }

31 \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
32 \cs_new_protected:Npn \@@_warning:n { \msg_warning:nn { nicematrix } }
33 \cs_new_protected:Npn \@@_warning:nnn { \msg_warning:nnnn { nicematrix } }
34 \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
35 \cs_generate_variant:Nn \@@_error:nn { n e }
36 \cs_new_protected:Npn \@@_error:nnnn { \msg_error:nnnn { nicematrix } }
37 \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
38 \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
39 \cs_new_protected:Npn \@@_msg_new:nn { \msg_new:nnn { nicematrix } }
40 \cs_new_protected:Npn \@@_critical:n { \msg_critical:nn { 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).

```

41 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
42 {
43   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
44   { \msg_new:nnn { nicematrix } { #1 } { #2 \ \ #3 } }
45   {
46     \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).

```

47     \msg_new:nnn { nicematrix } { #1~+ } { #3 }
48   }
49 }

```

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

```

50 \cs_new_protected:Npn \@@_error_or_warning:n
51 {
52   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
53   \@@_warning:n
54   \@@_error:n
55 }

```

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”.

```

56 \bool_new:N \g_@@_messages_for_Overleaf_bool
57 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
58 {
59   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
60   || \str_if_eq_p:ee \c_sys_jobname_str { output } % for Overleaf
61 }

```

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`.

```

62 \@@_msg_new:nn { class~revtex }
63 {
64   nicematrix~is~incompatible~with~REVTeX\\
65   You~can't~use~this~version~of~'nicematrix'~in~a~class~of~REVTeX~because~
66   REVTeX~is~*not*~compatible~with~recent~versions~of~LaTeX.~Sorry...
67   The~package~'nicematrix'~won't~be~loaded.
68 }
69 \IfClassLoadedT { revtex4-1 } { \@@_critical:n { class~revtex } }
70 \IfClassLoadedT { revtex4-2 } { \@@_critical:n { class~revtex } }

```

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

```

71 \cs_if_exist:NT \rvtx@ifformat@geq { \@@_critical:n { class~revtex } }

72 \@@_msg_new:nn { mdwtab~loaded }
73 {
74   'nicematrix'~is~incompatible~with~'mdwtab'\\
75   This~error~is~fatal.
76 }

77 \AtBeginDocument
78 { \IfPackageLoadedT { mdwtab } { \@@_fatal:n { mdwtab~loaded } } }

```

2 Collecting options

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

Example :

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

Therefore, by writing : `\def\G{\@@_collect_options:n{\F}}`,
 the command `\G` takes in an arbitrary number of optional arguments between square brackets.
 Be careful: that command is *not* “fully expandable” (because of `\peek_meaning:NTF`).

```

79 \cs_new_protected:Npn \@@_collect_options:n #1
80 {
81   \peek_meaning:NTF [
82     { \@@_collect_options:nw { #1 } }
83     { #1 { } }
84   }

```

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

```

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

```

3 Technical definitions

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

```

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

105 \tl_const:Nn \c_@@_brace_tl { nicematrix/brace }
106 \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).

```

107 \tl_new:N \l_@@_argspec_tl

108 \cs_generate_variant:Nn \seq_set_split:Nnn { N o }
109 \cs_generate_variant:Nn \str_set:Nn { N o }
110 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
111 \prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
112 \cs_generate_variant:Nn \dim_min:nn { v }
113 \cs_generate_variant:Nn \dim_max:nn { v }

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

```

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

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

```

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

```

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

```

126 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
127 {
128   \iow_now:Nn \@mainaux
129   {
130     \ExplSyntaxOn
131     \cs_if_free:NT \pgfsyspdfmark
132     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
133     \ExplSyntaxOff

```

```

134     }
135     \cs_gset_eq:NN \@_provide_pgfsyspdfmark: \prg_do_nothing:
136 }

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

172 \AtBeginDocument
173 {
174     \cs_set_protected:Npe \@@_everycr:
175     {
176         \IfPackageLoadedTF { colortbl } { \CT@everycr \everycr

```

```

177         { \noalign { \@@_in_everycr: } }
178     }
179     \IfPackageLoadedTF { colortbl }
180     {

```

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).

```

181         \cs_new_protected:Npn \@@_replace_columncolor:
182         {
183             \tl_replace_all:Nnn \l_@@_array_preamble_tl
184             \columncolor
185             \@@_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 color space.

```

186         }
187         \cs_new_eq:NN \@@_old_cellcolor: \cellcolor
188         \cs_new_eq:NN \@@_old_rowcolor: \rowcolor

```

Since the final user may use a `{tabular}` within a cell of an environment of `nicematrix`, we have to revert the definitions of the commands `\cellcolor`, `\rowcolor` and `\multicolumn` at the beginning of such environments `{tabular}`.

```

189         \hook_gput_code:nnn { env / tabular / begin } { \nicematrix }
190         {
191             \bool_if:NT \l_@@_in_env_bool
192             {
193                 \cs_set_eq:NN \cellcolor \@@_old_cellcolor:
194                 \cs_set_eq:NN \rowcolor \@@_old_rowcolor:
195                 \cs_set_eq:NN \multicolumn \@@_old_multicolumn:
196             }
197         }
198     }
199 }

```

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.

```

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

```

Idem for `\CT@drs@`.

```

207     \def \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
208     \def \CT@drs #1 #2
209     {
210         \dim_compare:nNt \baselineskip = \c_zero_dim { \noalign {
211             { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
212         }
213     }
214     \def \hline
215     {
216         \noalign { \ifnum 0 = ` } \fi
217         \cs_set_eq:NN \hskip \vskip
218         \cs_set_eq:NN \vrule \hrule
219         \cs_set_eq:NN \@width \@height
220         { \CT@arc@ \vline }
221         \futurelet \reserved@a
222         \@xhline
223     }
224     \cs_new_protected:Npn \@@_replace_columncolor:
225     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
226     \hook_gput_code:nnn { env / tabular / begin } { \nicematrix }

```

```

226     {
227         \bool_if:NT \l_@@_in_env_bool
228         { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
229     }
230 }
231 }
232 \cs_new_eq:NN \@@_old_multicolumn: \multicolumn

```

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.

```

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

```

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

```

243     \skip_horizontal:N \c_zero_dim
244 }

```

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`.

```

245     \everycr { }
246     \cr
247     \noalign { \skip_vertical:n { - \arrayrulewidth } }
248 }

```

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.

```

249 \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`.

```

250 { \@@_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*.

```

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

```

¹See question 99041 on TeX StackExchange.

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).

```

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

```

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

```

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

```

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

```

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

276 \cs_new_protected:Npn \@@_set_CTarc:n #1
277 {
278     \tl_if_blank:nF { #1 }
279     {
280         \tl_if_head_eq_meaning:nNTF { #1 } [
281             { \def \CT@arc@ { \color #1 } }
282             { \def \CT@arc@ { \color { #1 } } }
283         ]
284     }
285 \cs_generate_variant:Nn \@@_set_CTarc:n { o }

286 \cs_new_protected:Npn \@@_set_CTdrsc:n #1
287 {
288     \tl_if_head_eq_meaning:nNTF { #1 } [
289         { \def \CT@drsc@ { \color #1 } }
290         { \def \CT@drsc@ { \color { #1 } } }
291     ]

```

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

```

292 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
293 {
294     \tl_if_head_eq_meaning:nNTF { #2 } [
295         { #1 #2 }
296         { #1 { #2 } }
297     ]
298 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```

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

```

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

302 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
303 {
304     \tl_set_rescan:Nno
305     #1
306     {
307         \char_set_catcode_other:N >

```

```

308     \char_set_catcode_other:N <
309   }
310   #1
311 }

```

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

```

312 \dim_new:N \l_@@_tmpc_dim
313 \dim_new:N \l_@@_tmpd_dim
314 \tl_new:N \l_@@_tmpc_tl
315 \tl_new:N \l_@@_tmpd_tl
316 \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.

```

317 \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).

```

318 \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.

```

319 \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.

```

320 \box_new:N \l_@@_the_array_box

```

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

```

321 \cs_new_protected:Npn \@@_qpoint:n #1
322 { \pgfpointanchor { \@@_env: - #1 } { center } }

```

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

```

323 \bool_new:N \l_@@_tabular_bool

```

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

```

324 \bool_new:N \g_@@_delims_bool
325 \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.

```
326 \bool_new:N \l_@@_preamble_bool
327 \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.

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

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

```
329 \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.

```
330 \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).

```
331 \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.).

```
332 \dim_new:N \l_@@_col_width_dim
333 \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.

```
334 \int_new:N \g_@@_row_total_int
335 \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).

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

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

```
337 \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[l]{3cm}` will provide the value `l` for all the cells of the column.

```
338 \tl_new:N \l_@@_hpos_cell_tl
339 \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`.

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

Idem for the mono-row blocks.

```
341 \dim_new:N \g_@@_blocks_ht_dim
342 \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}`).

```
343 \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`.

```
344 \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.

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

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

```
346 \bool_new:N \l_@@_notes_detect_duplicates_bool
347 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

```
348 \bool_new:N \l_@@_initial_open_bool
349 \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.

```
350 \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).

```
351 \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).

```
352 \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).

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

The value of the key `color-of-false`

```
354 \tl_new:N \l_@@_color_of_false_tl
355 \tl_set:Nn \l_@@_color_of_false_tl { nocolor }
```

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
```

`\l_@@_bar_at_end_of_pream_bool` will be used for the construction of `\l_@@_array_preamble_tl`. It will be raised if you have a `|` at the end of the preamble provided by the final user and used during `\@@_create_col_nodes:`.

```
364 \bool_new:N \l_@@_bar_at_end_of_pream_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 }`).

```
365 \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).

```
366 \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.

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

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

```
369 \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}`).

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

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

```
371 \tl_new:N \l_@@_array_preamble_tl
```

For `\multicolumn`.

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

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

```
373 \tl_new:N \l_@@_pre_cell_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`.

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

```
375 \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 `:`.

```
376 \tl_new:N \l_@@_xdots_down_tl
377 \tl_new:N \l_@@_xdots_up_tl
378 \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`).

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

380 \cs_new_protected:Npn \@@_test_if_math_mode:
381 {
382   \if_mode_math: \else:
383     \@@_fatal:n { Outside-math-mode }
384   \fi:
385 }
```

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.

```
386 \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”.

```
387 \colorlet { nicematrix-last-col } { . }
388 \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`).

```
389 \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*.

```
390 \str_new:N \g_@@_com_or_env_str
391 \str_gset:Nn \g_@@_com_or_env_str { environment }
```

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

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

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

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

```
394 \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)`.

```
395 \cs_new:Npn \@@_full_name_env:
396 {
397   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
398     { command \space \c_backslash_str \g_@@_name_env_str }
399     { environment \space \{ \g_@@_name_env_str \} }
400 }

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

The argument is given by curryfication.

```
402 \cs_new_protected:Npn \@@_put_in_cell_after_hook:n
403 { \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.

```
404 \tl_new:N \g_@@_pre_code_before_tl
405 \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`.

```
406 \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.

```
407 \tl_new:N \g_@@_pre_code_after_tl
408 \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.

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

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

```
410 \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.

```
411 \int_new:N \l_@@_old_iRow_int
412 \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).

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

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

```
414 \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.

```
415 \bool_new:N \l_@@_X_columns_aux_bool
416 \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.

```
417 \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).

```
418 \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).

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

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

```
421 \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).

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

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

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

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

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

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

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

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

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

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

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

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

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

```
433 \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”.

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

```
435 \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.

```
436 \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!).

```
437 \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.

```
438 \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: $\{imin\}\{jmin\}\{imax\}\{jmax\}\{name\}$.

```
439 \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.

```
440 \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. In particular, it will be written on the file `aux`. However, for efficiency, the cells which are in a corner will also be marked directly in the main hash table of TeX.

```
441 \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`).

```
442 \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.

```
443 \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).

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

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

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

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

```
447 \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.

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

```
449 \int_new:N \g_@@_iddots_int
```

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

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

```

450 \dim_new:N \g_@@_delta_x_one_dim
451 \dim_new:N \g_@@_delta_y_one_dim
452 \dim_new:N \g_@@_delta_x_two_dim
453 \dim_new:N \g_@@_delta_y_two_dim

```

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

```

454 \int_new:N \l_@@_row_min_int
455 \int_new:N \l_@@_row_max_int
456 \int_new:N \l_@@_col_min_int
457 \int_new:N \l_@@_col_max_int

458 \int_new:N \l_@@_initial_i_int
459 \int_new:N \l_@@_initial_j_int
460 \int_new:N \l_@@_final_i_int
461 \int_new:N \l_@@_final_j_int

```

The following counters will be used when drawing the rules.

```

462 \int_new:N \l_@@_segment_start_int
463 \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.

```

464 \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).

```

465 \int_new:N \g_@@_static_num_of_col_int

```

The following parameters will store the “false columns” (with the letter F) in the preamble of the environment and the “false rows” created by `\FalseRow`.

```

466 \clist_new:N \g_@@_false_cols_clist
467 \clist_new:N \g_@@_false_rows_clist

```

The following parameters correspond to the keys `fill`, `opacity`, `draw`, `tikz`, `borders`, and `rounded-corners` of the command `\Block`.

```

468 \tl_new:N \l_@@_draw_tl
469 \seq_new:N \l_@@_tikz_seq
470 \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`).

```

471 \str_new:N \l_@@_hpos_block_str
472 \str_set:Nn \l_@@_hpos_block_str { c }
473 \bool_new:N \l_@@_hpos_of_block_cap_bool
474 \bool_new:N \l_@@_p_block_bool

475 \bool_new:N \l_@@_fix_vertex_bool

```

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

```
476 \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).

```
477 \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.

```
478 \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}`).

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

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

```
480 \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`).

```
481 \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.

```
482 \int_new:N \l_@@_first_row_int
483 \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.

```
484 \int_new:N \l_@@_first_col_int
485 \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).

```
486 \int_new:N \l_@@_last_row_int
487 \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

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

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

Idem for `\l_@@_last_col_without_value_bool`

```
489 \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 .

```
490 \int_new:N \l_@@_last_col_int
491 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

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

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

```
492 \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`).

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

Some utilities

```
494 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
495 {
```

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

```
496 \def \l_tmpa_tl { #1 }
497 \def \l_tmpb_tl { #2 }
498 }
```

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`.

```
499 \cs_new_protected:Npn \@@_expand_clist_hvlines:NN #1 #2
500 {
501 \clist_if_in:NnF #1 { all }
502 {
503 \clist_clear:N \l_tmpa_clist
504 \clist_map_inline:Nn #1
505 {
506 \tl_if_head_eq_meaning:nNTF { ##1 } -
507 {
```

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.

```

508         \int_if_zero:nF { #2 }
509         {
510             \clist_put_right:Ne \l_tmpa_clist
511             { \int_eval:n { #2 + (##1) + 1 } }
512         }
513     }
514     {

```

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

```

515         \tl_if_in:nnTF { ##1 } { - }
516         { \@@_cut_on_hyphen:w ##1 \q_stop }
517         {

```

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

```

518         \def \l_tmpa_tl { ##1 }
519         \def \l_tmpb_tl { ##1 }
520     }
521     \int_step_tokens:nnn \l_tmpa_tl \l_tmpb_tl
522     { \clist_put_right:Nn \l_tmpa_clist }
523 }
524 }
525 \tl_set_eq:NN #1 \l_tmpa_clist
526 }
527 }

```

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.

```

528 \AtBeginDocument
529 {
530     \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
531     \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
532 }

```

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 tabulernote}`. The first component is the optional argument (between square brackets) of the command `\tabulernote` (if the optional argument is not used, the value will be the special marker expressed by `\NoValue`).
- During the composition of the caption (value of `\l_@@_caption_tl`), the tabular notes will be numbered from 1 to `\g_@@_notes_caption_int` and the notes themselves will be stored in `\g_@@_notes_in_caption_seq`. The structure of the components of that sequence will be the same as for `\g_@@_notes_seq`.
- After the composition of the main tabular and after the composition of the caption, the sequences `\g_@@_notes_in_caption_seq` and `\g_@@_notes_seq` will be merged (in that order) and the notes will be composed.

The LaTeX counter `tabulernote` 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.

```
533 \newcounter { tabulernote }
```

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_@@_tabulernote_int`.

```
534 \int_new:N \g_@@_tabulernote_int
535 \cs_set:Npn \theHtabulernote { \int_use:N \g_@@_tabulernote_int }

536 \seq_new:N \g_@@_notes_seq
537 \seq_new:N \g_@@_notes_in_caption_seq
```

Before the actual tabular notes, it's possible to put a text specified by the key `tabulernote` of the environment. The token list `\g_@@_tabulernote_tl` corresponds to the value of that key.

```
538 \tl_new:N \g_@@_tabulernote_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.

```
539 \seq_new:N \l_@@_notes_labels_seq
540 \newcounter { nicematrix_draft }
541 \cs_new_protected:Npn \@@_notes_format:n #1
542 {
543   \setcounter { nicematrix_draft } { #1 }
544   \@@_notes_style:n { nicematrix_draft }
545 }
```

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

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

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

```
547 \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`.

```
548 \cs_new:Npn \@@_notes_label_in_list:n #1 { \textsuperscript { #1 } }
```

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

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`.

```
549 \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`).

```
550 \NewDocumentCommand { \tabularnote } { o m }
551 { \@@_err_enumitem_not_loaded: }

552 \AddToHook { package / enumitem / after }
553 {
```

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.

```
554 \newlist { tabularnotes } { enumerate } { 1 }
555 \setlist [ tabularnotes ]
556 {
557     topsep = \c_zero_dim ,
558     noitemsep ,
559     leftmargin = * ,
560     align = left ,
561     labelsep = \c_zero_dim ,
562     label =
563         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
564 }
565 \newlist { tabularnotes* } { enumerate* } { 1 }
566 \setlist [ tabularnotes* ]
567 {
568     afterlabel = \nobreak ,
569     itemjoin = \quad ,
570     label =
571         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
572 }
```

One must remind that we have allowed a `\tabularnote` 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`.

```
573 \RenewDocumentCommand { \tabularnote } { o m }
574 {
575     \bool_lazy_or:nnT \l_@@_in_env_bool { \cs_if_exist_p:N \@capytype }
576     {
577         \bool_lazy_and:nnTF \l_@@_in_env_bool { ! \l_@@_tabular_bool }
578         { \@@_error:n { tabularnote~forbidden } }
579     }
```

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

```
580 \bool_if:NTF \l_@@_in_caption_bool
581 {
582     \tl_if_novalue:nTF { #1 }
583     { \@@_tabularnote_capt:n { #1 } }
584     { \@@_tabularnote_capt:n { \exp_not:n { #1 } } }
585 }
586 {
587     \tl_if_novalue:nTF { #1 }
588     { \@@_tabularnote:n { #1 } }
589     { \@@_tabularnote:n { \exp_not:n { #1 } } }
```

```

590         }
591         { #2 }
592     }
593 }
594 }
595 }
596 \cs_new_protected:Npn \@@_err_enumitem_not_loaded:
597 {
598     \@@_error_or_warning:n { enumitem~not~loaded }
599     \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
600 }
601 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
602 { \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`.

```

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

```

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).

```

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

```

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`.

```

608     \int_zero:N \l_tmpb_int
609     \seq_map_indexed_inline:Nn \g_@@_notes_seq
610     {
611         \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
612         \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
613         {
614             \tl_if_novalue:nTF { #1 }
615             { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
616             { \int_set:Nn \l_tmpa_int { ##1 } }
617             \seq_map_break:
618         }
619     }
620     \int_if_zero:nF \l_tmpa_int
621     { \int_add:Nn \l_tmpa_int { \g_@@_notes_caption_int } }
622 }
623 \int_if_zero:nT \l_tmpa_int
624 {
625     \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
626     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
627 }
628 \seq_put_right:Ne \l_@@_notes_labels_seq
629 {
630     \tl_if_novalue:nTF { #1 }
631     {
632         \@@_notes_format:n

```

```

633         {
634             \int_eval:n
635             {
636                 \int_if_zero:nTF \l_tmpa_int
637                 \c@tabularnote
638                 \l_tmpa_int
639             }
640         }
641     }
642     { #1 }
643 }
644 \peek_meaning:NF \tabularnote
645 {

```

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`.

```

646     \hbox_set:Nn \l_tmpa_box
647     {

```

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

```

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

```

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

```

651     \int_gdecr:N \c@tabularnote
652     \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.

```

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

```

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

```

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`.

```

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

```

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

```

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

```

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

```

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.

```

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

```

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!

```

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

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

```

701 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
702 {
703   \begin { pgfscope }
704   \pgfset
705     {
706       inner~sep = \c_zero_dim ,
707       minimum~size = \c_zero_dim
708     }
709   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
710   \pgfnode
711     { rectangle }
712     { center }
713     {
714       \vbox_to_ht:nn

```

```

715         { \dim_abs:n { #5 - #3 } }
716     {
717         \vfill
718         \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
719     }
720 }
721 { #1 }
722 { }
723 \end { pgfscope }
724 }

```

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.

```

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

```

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).

```

748 \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.

```

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

```

752 \dim_new:N \l_@@_xdots_shorten_start_dim

```

```

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

```

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.

```

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

```

762 \tl_new:N \l_@@_xdots_line_style_tl
763 \tl_const:Nn \c_@@_standard_tl { standard }
764 \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`.

```

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

```

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

```

767 \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).

```

768 \bool_new:N \l_@@_auto_columns_width_bool
769 \bool_new:N \l_@@_row_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   color-of-false .tl_set:N = \l_@@_color_of_false_tl ,
870   color-of-false .value_required:n = true ,
871   fix-vertex .value_forbidden:n = true ,
872   brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
873   brace-shift+ .code:n =
874     \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
875   brace-shift+ .value_required:n = true ,
876   brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
877   caption-above .code:n = \@@_error_or_warning:n { caption-above~in~env } ,
878   show-cell-names .code = \@@_error_or_warning:n { show-cell-names } ,
879   color-inside .code:n = \@@_fatal:n { key~color~inside } ,
```

```

880   colortbl-like .code:n = \@@_fatal:n { key~color~inside } ,
881   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
882   &-in-blocks .meta:n = ampersand-in-blocks ,
883   no-cell-nodes .choices:nn = { true , false }
884   {
885     \tl_if_eq:NnTF \l_keys_choice_tl { true }
886     {
887       \bool_set_true:N \l_@@_no_cell_nodes_bool
888       \cs_set_eq:NN \@@_node_cell: \@@_node_cell_inactive:
889     }
890     {
891       \bool_set_false:N \l_@@_no_cell_nodes_bool
892       \cs_set_eq:NN \@@_node_cell: \@@_node_cell_active:
893     }
894   } ,
895   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.

```

896   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
897   rounded-corners .default:n = 4 pt ,
898   custom-line .code:n = \@@_custom_line:n { #1 } ,
899   default-line .code:n = \@@_default_line:n { #1 } ,
900   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
901   rules .value_required:n = true ,
902   trees .code:n = \keys_set:nn { nicematrix / trees } { #1 } ,
903   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`.

```

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

```

```

937 light-syntax-expanded .choices:nn = { true , false }
938 {
939     \tl_if_eq:NnTF \l_keys_value_tl { true }
940     {
941         \bool_set_true:N \l_@@_light_syntax_bool
942         \bool_set_true:N \l_@@_light_syntax_expanded_bool
943     }
944     {
945         \bool_set_false:N \l_@@_light_syntax_bool
946         \bool_set_false:N \l_@@_light_syntax_expanded_bool
947     }
948 } ,
949 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.

```

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

```

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

```

998         { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
999     }
1000     { \@@_error:n { One~letter~allowed } }
1001 },
1002 vl_lines-in-sub-matrix .value_required:n = true ,
1003 hv_lines .code:n =
1004 {
1005     \bool_set_true:N \l_@@_hv_lines_bool
1006     \tl_set_eq:NN \l_@@_vl_lines_clist \c_@@_all_tl
1007     \tl_set_eq:NN \l_@@_hl_lines_clist \c_@@_all_tl
1008 },
1009 hv_lines .value_forbidden:n = true ,
1010 hv_lines-except-borders .code:n =
1011 {
1012     \tl_set_eq:NN \l_@@_vl_lines_clist \c_@@_all_tl
1013     \tl_set_eq:NN \l_@@_hl_lines_clist \c_@@_all_tl
1014     \bool_set_true:N \l_@@_hv_lines_bool
1015     \bool_set_true:N \l_@@_except_borders_bool
1016 },
1017 hl_lines-except-borders .value_forbidden:n = true ,
1018 hl_lines-except-borders .code:n =
1019 {
1020     \tl_set_eq:NN \l_@@_hl_lines_clist \c_@@_all_tl
1021     \bool_set_true:N \l_@@_hl_lines_bool
1022     \bool_set_true:N \l_@@_except_borders_bool
1023 },
1024 hl_lines-except-borders .value_forbidden:n = true ,
1025 parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,
1026 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.

```

1027 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
1028 renew-dots .value_forbidden:n = true ,
1029 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
1030 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
1031 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
1032 create-extra-nodes .meta:n =
1033 { create-medium-nodes , create-large-nodes } ,
1034 left-margin .dim_set:N = \l_@@_left_margin_dim ,
1035 left-margin .default:n = \arraycolsep ,
1036 right-margin .dim_set:N = \l_@@_right_margin_dim ,
1037 right-margin .default:n = \arraycolsep ,
1038 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
1039 margin .default:n = \arraycolsep ,
1040 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
1041 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
1042 extra-margin .meta:n =
1043 { extra-left-margin = #1 , extra-right-margin = #1 } ,
1044 extra-margin .value_required:n = true ,
1045 respect-arraystretch .code:n =
1046 \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
1047 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.

```

1048 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
1049 pgf-node-code .value_required:n = true ,
1050 }

```

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

```

1051 \keys_define:nn { nicematrix / environments }
1052 {
1053   draw-trees-in-col .code:n =
1054     \clist_gput_right:Nn \g_@@_col_with_trees_clist { #1 } ,
1055   draw-trees-in-col .value_required:n = true ,
1056   create-blocks-in-col .code:n =
1057     \clist_gput_right:Nn \g_@@_cbic_clist { #1 } ,
1058   create-blocks-in-col .value_required:n = true ,
1059   corners .clist_set:N = \l_@@_corners_clist ,
1060   corners .default:n = { NW , SW , NE , SE } ,
1061   code-before .code:n =
1062     {
1063       \tl_if_empty:nF { #1 }
1064       {
1065         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1066         \bool_set_true:N \l_@@_code_before_bool
1067       }
1068     } ,
1069   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}`.

```

1070   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
1071   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
1072   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).

```

1073   baseline .tl_set:N = \l_@@_baseline_tl ,
1074   baseline .value_required:n = true ,
1075   baseline .initial:n = c ,
1076   columns-width .code:n =
1077     \str_case:nnF { #1 }
1078     {
1079       { auto } { \bool_set_true:N \l_@@_auto_columns_width_bool }
1080       { row-height } { \bool_set_true:N \l_@@_row_columns_width_bool }
1081     }
1082     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
1083   columns-width .value_required:n = true ,
1084   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.

```

1085     \legacy_if:nF { measuring@ }
1086     {
1087       \str_set:Ne \l_@@_name_str { #1 }
1088       \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
1089       { \@@_err_duplicate_names:n { #1 } }
1090       { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
1091     } ,
1092   name .value_required:n = true ,
1093   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
1094   code-after .value_required:n = true ,
1095 }

1096 \cs_set:Npn \@@_err_duplicate_names:n #1
1097 { \@@_error:nn { Duplicate~name } { #1 } }

1098 \keys_define:nn { nicematrix / notes }
1099 {
1100   no-print .bool_set:N = \l_@@_notes_no_print_bool ,
1101   para .bool_set:N = \l_@@_notes_para_bool ,
1102   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
1103   code-before .value_required:n = true ,

```

```

1104 code-before+ .code:n =
1105   \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1106 code-before+ .value_required:n = true ,
1107 code-before~+ .code:n =
1108   \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1109 code-before~+ .value_required:n = true ,
1110 code-after .tl_set:N = \l_@@_notes_code_after_tl ,
1111 code-after .value_required:n = true ,
1112 bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
1113 style .cs_set:Np = \@@_notes_style:n #1 ,
1114 style .value_required:n = true ,
1115 label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
1116 label-in-tabular .value_required:n = true ,
1117 label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1118 label-in-list .value_required:n = true ,
1119 enumitem-keys .code:n =
1120   {
1121     \AtBeginDocument
1122     {
1123       \IfPackageLoadedT { enumitem }
1124       { \setlist* [ tabularnotes ] { #1 } }
1125     }
1126   } ,
1127 enumitem-keys .value_required:n = true ,
1128 enumitem-keys-para .code:n =
1129   {
1130     \AtBeginDocument
1131     {
1132       \IfPackageLoadedT { enumitem }
1133       { \setlist* [ tabularnotes* ] { #1 } }
1134     }
1135   } ,
1136 enumitem-keys-para .value_required:n = true ,
1137 detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1138 unknown .code:n =
1139   \@@_unknown_key:nn { nicematrix / notes } { Unknown~key~for~notes }
1140 }

```

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.

```

1141 \keys_define:nn { nicematrix / delimiters }
1142 {
1143   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1144   color .tl_set:N = \l_@@_delimiters_color_tl ,
1145   color .value_required:n = true ,
1146 }

```

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

```

1147 \keys_define:nn { nicematrix }
1148 {
1149   NiceMatrixOptions .inherit:n =
1150     { nicematrix / Global } ,
1151   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1152   NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1153   NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1154   NiceMatrixOptions / trees .inherit:n = nicematrix / trees ,

```

```

1155 NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1156 SubMatrix / rules .inherit:n = nicematrix / rules ,
1157 CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1158 CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1159 CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1160 NiceMatrix .inherit:n =
1161 {
1162     nicematrix / Global ,
1163     nicematrix / environments ,
1164 } ,
1165 NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1166 NiceMatrix / rules .inherit:n = nicematrix / rules ,
1167 NiceMatrix / trees .inherit:n = nicematrix / trees ,
1168 NiceTabular .inherit:n =
1169 {
1170     nicematrix / Global ,
1171     nicematrix / environments
1172 } ,
1173 NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1174 NiceTabular / rules .inherit:n = nicematrix / rules ,
1175 NiceTabular / notes .inherit:n = nicematrix / notes ,
1176 NiceTabular / trees .inherit:n = nicematrix / trees ,
1177 NiceArray .inherit:n =
1178 {
1179     nicematrix / Global ,
1180     nicematrix / environments ,
1181 } ,
1182 NiceArray / xdots .inherit:n = nicematrix / xdots ,
1183 NiceArray / rules .inherit:n = nicematrix / rules ,
1184 NiceArray / trees .inherit:n = nicematrix / trees ,
1185 pNiceArray .inherit:n =
1186 {
1187     nicematrix / Global ,
1188     nicematrix / environments ,
1189 } ,
1190 pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1191 pNiceArray / rules .inherit:n = nicematrix / rules ,
1192 pNiceArray / trees .inherit:n = nicematrix / trees
1193 }

```

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

```

1194 \keys_define:nn { nicematrix / NiceMatrixOptions }
1195 {
1196     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1197     delimiters / color .value_required:n = true ,
1198     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1199     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1200     delimiters .value_required:n = true ,
1201     width .dim_set:N = \l_@@_width_dim ,
1202     last-col .code:n =
1203     \tl_if_empty:nF { #1 }
1204     { \@@_error:n { last-col~non-empty~for~NiceMatrixOptions } }
1205     \int_zero:N \l_@@_last_col_int ,
1206     small .bool_set:N = \l_@@_small_bool ,
1207     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.

```

1208     renew-matrix .code:n = \@@_renew_matrix: ,
1209     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}`.

```
1210     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.

```
1211     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)`.

```
1212     \str_if_eq:eeTF { #1 } { auto }
1213     { \@@_error:n { Option~auto~for~columns~width } }
1214     { \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.

```
1215     allow-duplicate-names .code:n =
1216     \cs_set:Nn \@@_err_duplicate_names:n { } ,
1217     allow-duplicate-names .value_forbidden:n = true ,
1218     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1219     notes .value_required:n = true ,
1220     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1221     sub-matrix .value_required:n = true ,
1222     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1223     matrix / columns-type .value_required:n = true ,
1224     caption-above .bool_set:N = \l_@@_caption_above_bool ,
1225     unknown .code:n =
1226     \@@_unknown_key:nn
1227     { nicematrix / Global , nicematrix / NiceMatrixOptions }
1228     { Unknown-key-for-NiceMatrixOptions }
1229 }
```

`\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.

```
1230 \NewDocumentCommand \NiceMatrixOptions { m }
1231 { \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.

```
1232 \keys_define:nn { nicematrix / NiceMatrix }
1233 {
1234     last-col .code:n = \tl_if_empty:nTF { #1 }
1235     {
1236         \bool_set_true:N \l_@@_last_col_without_value_bool
1237         \int_set:Nn \l_@@_last_col_int { -1 }
1238     }
1239     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1240     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1241     columns-type .value_required:n = true ,
1242     l .meta:n = { columns-type = l } ,
1243     r .meta:n = { columns-type = r } ,
1244     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1245     delimiters / color .value_required:n = true ,
1246     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1247     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1248     delimiters .value_required:n = true ,
1249     small .bool_set:N = \l_@@_small_bool ,
1250     small .value_forbidden:n = true ,
1251     unknown .code:n =
```

```

1252 \@@_unknown_key:nn
1253 { nicematrix / Global , nicematrix / environments , nicematrix / NiceMatrix }
1254 { Unknown-key-for-NiceMatrix }
1255 }

```

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

```

1256 \keys_define:nn { nicematrix / NiceArray }
1257 {

```

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.

```

1258 small .bool_set:N = \l_@@_small_bool ,
1259 small .value_forbidden:n = true ,
1260 last-col .code:n = \tl_if_empty:nF { #1 }
1261 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1262 \int_zero:N \l_@@_last_col_int ,
1263 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1264 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1265 unknown .code:n =
1266 \@@_unknown_key:nn
1267 { nicematrix / NiceArray , nicematrix / Global , nicematrix / environments }
1268 { Unknown-key-for-NiceArray }
1269 }

1270 \keys_define:nn { nicematrix / pNiceArray }
1271 {
1272 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1273 last-col .code:n = \tl_if_empty:nF { #1 }
1274 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1275 \int_zero:N \l_@@_last_col_int ,
1276 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1277 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1278 delimiters / color .value_required:n = true ,
1279 delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1280 delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1281 delimiters .value_required:n = true ,
1282 small .bool_set:N = \l_@@_small_bool ,
1283 small .value_forbidden:n = true ,
1284 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1285 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1286 unknown .code:n =
1287 \@@_unknown_key:nn
1288 { nicematrix / pNiceArray , nicematrix / Global , nicematrix / environments }
1289 { Unknown-key-for-NiceMatrix }
1290 }

```

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

```

1291 \keys_define:nn { nicematrix / NiceTabular }
1292 {

```

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.

```

1293 width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1294 \bool_set_true:N \l_@@_width_used_bool ,
1295 width .value_required:n = true ,
1296 notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1297 tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1298 tabularnote .value_required:n = true ,
1299 caption .tl_set:N = \l_@@_caption_tl ,
1300 caption .value_required:n = true ,

```

```

1301 short-caption .tl_set:N = \l_@@_short_caption_tl ,
1302 short-caption .value_required:n = true ,
1303 label .tl_set:N = \l_@@_label_tl ,
1304 label .value_required:n = true ,
1305 last-col .code:n = \tl_if_empty:nF { #1 }
1306 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1307 \int_zero:N \l_@@_last_col_int ,
1308 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1309 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1310 unknown .code:n =
1311 \@@_unknown_key:nn
1312 { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
1313 { Unknown-key-for-NiceTabular }
1314 }

```

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
```

```

1315 \keys_define:nn { nicematrix / CodeAfter }
1316 {
1317 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1318 delimiters / color .value_required:n = true ,
1319 rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1320 rules .value_required:n = true ,
1321 xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1322 sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1323 sub-matrix .value_required:n = true ,
1324 unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1325 }

```

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}`).

```

1326 \cs_new_protected:Npn \@@_cell_begin:
1327 {

```

`\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.

```
1328 \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).

```
1329 \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.

```

1330 \cs_set_eq:NN \Hline \@@_Hline_in_cell:
1331 \cs_set_eq:NN \FalseRow \@@_FalseRow_in_cell:

```

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

```
1332 \int_gincr:N \c@jCol
```

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

Here is a version with the standard syntax of L3.

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

We will use a version a little more efficient.

```
1333   \if_int_compare:w \c@jCol = \c_one_int
1334   \if_int_compare:w \l_@@_first_col_int = \c_one_int
1335   \@@_begin_of_row:
1336   \fi:
1337   \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:`.

```
1338   \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1339   \@@_tuning_not_tabular_begin:
1340   \@@_tuning_exterior_rows:
1341   \g_@@_row_style_tl
1342   }
1343   \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.

```
1344 \cs_new_protected:Npn \@@_tuning_first_last_row:
1345 {
1346   \if_int_compare:w \c@iRow = \c_zero_int
1347   \if_int_compare:w \c@jCol > \c_zero_int
1348   \l_@@_code_for_first_row_tl
1349   \@@_tuning_key_small:
1350   \xglobal \colorlet { nicematrix-first-row } { . }
1351   \fi:
1352   \else:
1353   \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row:
1354   \fi:
1355 }
```

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.

```

1356 \cs_new_protected:Npn \@@_tuning_last_row:
1357 {
1358   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1359     \l_@@_code_for_last_row_tl
1360     \@@_tuning_key_small:
1361     \xglobal \colorlet { nicematrix-last-row } { . }
1362   \fi:
1363 }

```

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

```

1364 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1365 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1366 {
1367   \m@th
1368   $ % $

```

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

```

1369   \@@_tuning_key_small:
1370 }
1371 \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.

```

1372 \cs_new_protected:Npn \@@_begin_of_row:
1373 {
1374   \int_gincr:N \c@iRow
1375   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1376   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1377   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1378   \pgfpicture
1379   \pgfrememberpicturepositiononpagetrue
1380   \pgfcoordinate
1381     { \@@_env: - row - \int_use:N \c@iRow - base }
1382     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1383   \str_if_empty:NF \l_@@_name_str
1384     {
1385       \pgfnodealias
1386         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1387         { \@@_env: - row - \int_use:N \c@iRow - base }
1388     }
1389   \endpgfpicture
1390 }

```

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.

```

1391 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1392 {
1393   \if_int_compare:w \c@iRow = \c_zero_int
1394     \if_dim:w \box_dp:N \l_@@_cell_box > \g_@@_dp_row_zero_dim
1395       \global \g_@@_dp_row_zero_dim = \box_dp:N \l_@@_cell_box
1396     \fi:
1397     \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_zero_dim
1398       \global \g_@@_ht_row_zero_dim = \box_ht:N \l_@@_cell_box
1399     \fi:
1400   \else:
1401     \if_int_compare:w \c@iRow = \c_one_int
1402       \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_one_dim
1403         \global \g_@@_ht_row_one_dim = \box_ht:N \l_@@_cell_box
1404       \fi:
1405     \fi:
1406   \fi:
1407 }

1408 \cs_new_protected:Npn \@@_rotate_cell_box:
1409 {
1410   \box_rotate:Nn \l_@@_cell_box
1411   { \bool_if:NTF \g_@@_rotate_minus_bool { -90 } { 90 } }
1412   \bool_if:NTF \g_@@_rotate_c_bool
1413   {
1414     \hbox_set:Nn \l_@@_cell_box
1415     { \vbox_center:n { \box_use:N \l_@@_cell_box } }
1416   }
1417   {
1418     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1419     {
1420       \vbox_set_top:Nn \l_@@_cell_box
1421       {
1422         \vbox_to_zero:n { }
1423         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1424         \box_use:N \l_@@_cell_box
1425       }
1426     }
1427   }
1428   \bool_gset_false:N \g_@@_rotate_bool
1429   \bool_gset_false:N \g_@@_rotate_c_bool
1430   \bool_gset_false:N \g_@@_rotate_minus_bool
1431 }

```

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.

```

1432 \cs_set_protected:Npn \@@_adjust_size_box:
1433 {
1434     \if_dim:w \g_@@_blocks_wd_dim > \c_zero_dim
1435     \if_dim:w \g_@@_blocks_wd_dim > \box_wd:N \l_@@_cell_box
1436     \box_wd:N \l_@@_cell_box = \g_@@_blocks_wd_dim
1437     \fi:
1438     \global \g_@@_blocks_wd_dim = \c_zero_dim
1439     \fi:
1440     \if_dim:w \g_@@_blocks_dp_dim > \c_zero_dim
1441     \if_dim:w \g_@@_blocks_dp_dim > \box_dp:N \l_@@_cell_box
1442     \box_dp:N \l_@@_cell_box = \g_@@_blocks_dp_dim
1443     \fi:
1444     \global \g_@@_blocks_dp_dim = \c_zero_dim
1445     \fi:
1446     \if_dim:w \g_@@_blocks_ht_dim > \c_zero_dim
1447     \if_dim:w \g_@@_blocks_ht_dim > \box_ht:N \l_@@_cell_box
1448     \box_ht:N \l_@@_cell_box = \g_@@_blocks_ht_dim
1449     \fi:
1450     \global \g_@@_blocks_ht_dim = \c_zero_dim
1451     \fi:
1452 }
1453 \cs_new_protected:Npn \@@_cell_end:
1454 {

```

The following command is nullified in the tabulars.

```

1455     \@@_tuning_not_tabular_end:
1456     \hbox_set_end:
1457     \@@_cell_end_i:
1458 }

```

```

\cs_new_protected:Npn \@@_cell_end_i:
{
    \g_@@_cell_after_hook_tl
    \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
    \@@_adjust_size_box:
    \box_set_ht:Nn \l_@@_cell_box
    { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
    \box_set_dp:Nn \l_@@_cell_box
    { \box_dp:N \l_@@_cell_box + \l_@@_cell_space_bottom_limit_dim }
    \@@_update_max_cell_width:
    \@@_update_for_first_and_last_row:
    \bool_if:NTF \g_@@_empty_cell_bool
    { \box_use_drop:N \l_@@_cell_box }
    {
        \bool_if:NTF \g_@@_not_empty_cell_bool
        { \@@_print_node_cell: }
    }
}

```

```

    {
      \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
      { \@@_print_node_cell: }
      { \box_use_drop:N \l_@@_cell_box }
    }
  }
\int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
{ \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
\bool_gset_false:N \g_@@_empty_cell_bool
\bool_gset_false:N \g_@@_not_empty_cell_bool
}

```

Here is a version slightly more efficient.

```

1459 \cs_new_protected:Npn \@@_cell_end_i:
1460 {

```

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.

```

1461   \g_@@_cell_after_hook_tl
1462   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1463   \@@_adjust_size_box:
1464   \box_set_ht:Nn \l_@@_cell_box
1465   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1466   \box_set_dp:Nn \l_@@_cell_box
1467   { \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 potential “first column” and the potential “last column”).

```

1468   \@@_update_max_cell_width:

```

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

```

1469   \@@_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.

```

1470   \bool_if:NNTF \g_@@_empty_cell_bool
1471   { \box_use_drop:N \l_@@_cell_box }
1472   {
1473     \bool_if:NNTF \g_@@_not_empty_cell_bool
1474     \@@_print_node_cell:
1475     {
1476       \if_dim:w \box_wd:N \l_@@_cell_box > \c_zero_dim
1477       \@@_print_node_cell:
1478       \else:

```

```

1479         \box_use_drop:N \l_@@_cell_box
1480     \fi:
1481 }
1482 }
1483 \if_int_compare:w \c@jCol > \g_@@_col_total_int
1484     \global \g_@@_col_total_int = \c@jCol
1485 \fi:
1486 \global \let \g_@@_empty_cell_bool \c_false_bool
1487 \global \let \g_@@_not_empty_cell_bool \c_false_bool
1488 }

```

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:

```

1489 \cs_new_protected:Npn \@@_update_max_cell_width:
1490 {
1491     \if_dim:w \box_wd:N \l_@@_cell_box > \g_@@_max_cell_width_dim
1492         \global \g_@@_max_cell_width_dim = \box_wd:N \l_@@_cell_box
1493     \fi:
1494 }

```

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`).

```

1495 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1496 {
1497     \@@_math_toggle:
1498     \hbox_set_end:
1499     \bool_if:NF \g_@@_rotate_bool
1500     {
1501         \hbox_set:Nn \l_@@_cell_box
1502         {
1503             \makebox [ \l_@@_col_width_dim ] [ s ]
1504             { \hbox_unpack_drop:N \l_@@_cell_box }
1505         }
1506     }
1507     \@@_cell_end_i:
1508 }

```

```

1509 \pgfset
1510 {
1511     nicematrix / cell-node /.style =
1512     {
1513         inner~sep = \c_zero_dim ,
1514         minimum~width = \c_zero_dim
1515     }
1516 }

```

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.

```

1517 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1518 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1519 {

```

```

1520 \use:c
1521 {
1522   __siunitx_table_align_
1523   \bool_if:NTF \l__siunitx_table_text_bool
1524   \l__siunitx_table_align_text_tl
1525   \l__siunitx_table_align_number_str
1526   :n
1527 }
1528 { #1 }
1529 }

```

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!

```

1530 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1531 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1532 {
1533   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1534   \hbox:n
1535   {
1536     \pgfsys@markposition
1537     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1538   }
1539   #1
1540   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1541   \hbox:n
1542   {
1543     \pgfsys@markposition
1544     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1545   }
1546 }

1547 \cs_new_protected:Npn \@@_print_node_cell:
1548 {
1549   \socket_use:nn { nicematrix / siunitx-wrap }
1550   { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1551 }

```

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

```

1552 \cs_new_protected:Npn \@@_node_cell_active:
1553 {
1554   \pgfpicture
1555   \pgfsetbaseline \c_zero_dim
1556   \pgfrememberpicturepositiononpagetrue
1557   \pgfset { nicematrix / cell-node }
1558   \pgfnode
1559   { rectangle }
1560   { base }
1561   {

```

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).

```

1562   \sys_if_engine_xetex:T { \set@color }
1563   \box_use:N \l_@@_cell_box
1564 }
1565 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1566 { \l_@@_pgf_node_code_tl }
1567 \str_if_empty:NF \l_@@_name_str
1568 {
1569   \pgfnodealias

```

```

1570      { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1571      { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1572    }
1573    \endpgfpicture
1574  }
1575  \cs_set_eq:NN \@@_node_cell: \@@_node_cell_active:
1576  \cs_new_protected:Npn \@@_node_cell_inactive:
1577    { \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).

```

1578 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1579 {
1580   \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce
1581     { g_@@_ #2 _ lines _ tl }
1582   {
1583     \use:c { @@ _ draw _ #2 : nnn }
1584     { \int_use:N \c@iRow }
1585     { \int_use:N \c@jCol }
1586     { \exp_not:n { #3 } }
1587   }
1588 }

1589 \cs_new_protected:Npn \@@_array:n
1590 {
1591   \dim_set:Nn \col@sep
1592   {
1593     \bool_if:NTF \l_@@_row_columns_width_bool
1594       { \c_zero_dim }
1595       { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1596   }
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. Re-
mark that \str_if_eq:eeTF is fully expandable and we need something fully expandable here.
\str_if_eq:ee(TF) is faster than \str_if_eq:nn(TF).
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:nNt \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:NeT
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 \cs_new_protected:Npn \@@_patch_booktabs: { }
1678 \AddToHook { package / booktabs / after }
1679 {
1680     \cs_set_protected:Npn \@@_patch_booktabs:
1681         { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1682 }

```

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`.

```

1683 \cs_new_protected:Npn \@@_some_initialization:
1684 {
1685     \@@_everycr:
1686     \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1687     \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1688     \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1689     \dim_gzero:N \g_@@_dp_ante_last_row_dim
1690     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1691     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1692 }

```

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`.

```
1693 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1694 {
```

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

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

```
1695 \seq_gclear:N \g_@@_pos_of_blocks_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.

```
1696 \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.

```
1697 \fp_gzero:N \g_@@_total_X_weight_fp
1698 \bool_gset_false:N \g_@@_V_of_X_bool

1699 \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1700 \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol

1701 \@@_patch_booktabs:
1702 \box_clear_new:N \l_@@_cell_box
1703 \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}`).

```
1704 \bool_if:NT \l_@@_small_bool
1705 {
1706     \def \arraystretch { 0.47 }
1707     \dim_set:Nn \arraycolsep { 1.45 pt }
```

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

```
1708 \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1709 }
```

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!

```
1710 \bool_if:NT \g_@@_create_cell_nodes_bool
1711 {
1712     \tl_put_right:Nn \@@_begin_of_row:
1713     {
1714         \pgfsys@markposition
1715         { \@@_env: - row - \int_use:N \c@iRow - base }
1716     }
1717     \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1718 }
```

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`.

```

1719   \def \ar@ialign
1720   {
1721     \tbl_init_cell_data_for_table:
1722     \@@_some_initialization:
1723     \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`.

```

1724     \cs_set_eq:Nc \ar@ialign { \@@_old_ar@ialign: }
1725     \halign
1726   }

```

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).

```

1727   \cs_set_eq:NN \@@_old_ldots: \ldots
1728   \cs_set_eq:NN \@@_old_cdots: \cdots
1729   \cs_set_eq:NN \@@_old_vdots: \vdots
1730   \cs_set_eq:NN \@@_old_ddots: \ddots
1731   \cs_set_eq:NN \@@_old_iddots: \iddots
1732   \bool_if:NTF \l_@@_standard_cline_bool
1733   { \cs_set_eq:NN \cline \@@_standard_cline: }
1734   { \cs_set_eq:NN \cline \@@_cline: }
1735   \bool_if:NTF \l_@@_no_cell_nodes_bool
1736   { \@@_redefine_xdots_no_cell_nodes: }
1737   { \@@_redefine_xdots: }
1738   \cs_set_eq:NN \Hline \@@_Hline:
1739   \cs_set_eq:NN \Hspace \@@_Hspace:
1740   \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1741   \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1742   \cs_set_eq:NN \Block \@@_Block:
1743   \cs_set_eq:NN \rotate \@@_rotate:
1744   \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1745   \cs_set_eq:NN \dotfill \@@_dotfill:
1746   \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1747   \cs_if_free:NT \Body { \cs_set_eq:NN \Body \@@_Body: }
1748   \cs_if_free:NT \CodeBefore { \cs_set_eq:NN \CodeBefore \@@_CodeBefore: }
1749   \cs_set_eq:NN \diagbox \@@_diagbox:nn
1750   \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1751   \cs_set_eq:NN \TopRule \@@_TopRule
1752   \cs_set_eq:NN \MidRule \@@_MidRule
1753   \cs_set_eq:NN \BottomRule \@@_BottomRule
1754   \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1755   \cs_set_eq:NN \Hbrace \@@_Hbrace
1756   \cs_set_eq:NN \Vbrace \@@_Vbrace
1757   \cs_set_eq:NN \hdashedline \@@_hdashedline:
1758   \cs_set_eq:NN \cdashedline \@@_cdashedline:n
1759   \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1760   { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1761   \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1762   \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1763   \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1764   \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1765   \cs_set_eq:NN \FalseRow \@@_FalseRow
1766   \int_if_zero:nTF \l_@@_first_row_int
1767   { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_first_last_row: }
1768   {
1769     \int_compare:nNnT \l_@@_last_row_int > \c_zero_int

```

```

1770      { \cs_gset_eq:NN \@_tuning_exterior_rows: \@_tuning_last_row: }
1771    }
1772    \bool_if:NT \l_@@_renew_dots_bool { \@_renew_dots: }

```

We redefine `\multicolumn`⁷.

```

1773    \cs_set_eq:NN \multicolumn \@_multicolumn:nnn

```

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

```

1774    \tl_if_exist:NT \l_@@_note_in_caption_tl
1775    {
1776      \tl_if_empty:NF \l_@@_note_in_caption_tl
1777      {
1778        \int_gset:Nn \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1779        \int_gset:Nn \c@tabularnote \l_@@_note_in_caption_tl
1780      }
1781    }

```

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).

```

1782    \seq_gclear:N \g_@@_multicolumn_cells_seq
1783    \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`).

```

1784    \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).

```

1785    \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`).

```

1786    \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.

```

1787    \int_gzero:N \g_@@_col_total_int
1788    \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1789    \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.

```

1790    \tl_gclear_new:N \g_@@_Cdots_lines_tl
1791    \tl_gclear_new:N \g_@@_Ldots_lines_tl
1792    \tl_gclear_new:N \g_@@_Vdots_lines_tl
1793    \tl_gclear_new:N \g_@@_Ddots_lines_tl
1794    \tl_gclear_new:N \g_@@_Iddots_lines_tl
1795    \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

1796    \tl_gclear:N \g_nicematrix_code_before_tl
1797    \tl_gclear:N \g_@@_pre_code_before_tl

```

⁷Since we want `\multicolumn` to be available in the potential environments `{tabular}` nested in the environments of `nicematrix`, we have used the hook `tabular/begin` to revert the definition.

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]`).

```

1798 \dim_zero_new:N \l_@@_left_delim_dim
1799 \dim_zero_new:N \l_@@_right_delim_dim
1800 \bool_if:NTF \g_@@_delims_bool
1801 {

```

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

```

1802 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1803 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1804 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1805 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1806 }
1807 {
1808 \dim_gset:Nn \l_@@_left_delim_dim
1809 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1810 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1811 }
1812 }

```

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.

```

1813 \cs_new_protected:Npn \@@_pre_array:
1814 {
1815 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1816 \int_gzero_new:N \c@iRow
1817 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1818 \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).

```

1819 \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1820 { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1821 \int_compare:nNnT \l_@@_last_col_int > \c_zero_int
1822 { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1823 \bool_if:NT \g_@@_aux_found_bool
1824 {
1825 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1826 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1827 \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1828 \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1829 }

```

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).

```

1830 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1831 {
1832 \bool_set_true:N \l_@@_last_row_without_value_bool
1833 \bool_if:NT \g_@@_aux_found_bool
1834 { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1835 }
1836 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1837 {
1838 \bool_if:NT \g_@@_aux_found_bool

```

```

1839      { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1840    }

```

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”.

```

1841    \int_compare:nNtT \l_@@_last_row_int > { -2 }
1842    {
1843      \tl_put_right:Nn \@@_update_for_first_and_last_row:
1844      {
1845        \dim_compare:nNtT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1846        { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1847        \dim_compare:nNtT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1848        { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1849      }
1850    }

1851    \seq_gclear:N \g_@@_cols_vlism_seq
1852    \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1853    \bool_if:NT \l_@@_code_before_bool \@@_exec_code_before:

```

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

```

1854    \@@_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).

```

1855    \hbox_set:Nw \l_@@_the_array_box
1856    \skip_horizontal:N \l_@@_left_margin_dim
1857    \skip_horizontal:N \l_@@_extra_left_margin_dim
1858    \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).

```

1859    \m@th
1860    $ % $
1861    \bool_if:NtF \l_@@_light_syntax_bool
1862    { \use:c { @@-light-syntax } }
1863    { \use:c { @@-normal-syntax } }
1864  }

```

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

```

1865    \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1866    {
1867      \tl_set:Nn \l_tmpa_tl { #1 }
1868      \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1869      { \@@_rescan_for_spanish:N \l_tmpa_tl }
1870      \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1871      \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...

```

1872    \@@_pre_array:
1873  }

```

```

1874 \cs_new_protected:Npn \@@_redefine_xdots:
1875 {
1876   \cs_set_eq:NN \Ldots \@@_Ldots:
1877   \cs_set_eq:NN \Cdots \@@_Cdots:
1878   \cs_set_eq:NN \Vdots \@@_Vdots:
1879   \cs_set_eq:NN \Ddots \@@_Ddots:
1880   \cs_set_eq:NN \Iddots \@@_Iddots:
1881 }
1882 \cs_new_protected:Npn \@@_redefine_xdots_no_cell_nodes:
1883 {
1884   \cs_set_protected:Npn \Ldots { \@@_error_xdots:N \Ldots }
1885   \cs_set_protected:Npn \Cdots { \@@_error_xdots:N \Cdots }
1886   \cs_set_protected:Npn \Vdots { \@@_error_xdots:N \Vdots }
1887   \cs_set_protected:Npn \Ddots { \@@_error_xdots:N \Ddots }
1888   \cs_set_protected:Npn \Iddots { \@@_error_xdots:N \Iddots }
1889 }
1890 \cs_new_protected:Npn \@@_error_xdots:N #1
1891 { \@@_error:nn { xdots~without~cell~nodes} { #1 } }

```

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:nnn \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:nnn \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:nNt { \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
```

The following `\scan_stop`: is a defensive programming for the case where the final user has written in the `\CodeBefore` something like `\rowcolor{red!15}` (without the second mandatory argument which should be a list of numbers of rows).

```
1964 \scan_stop:
```

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.

```
1965 \@@_actually_color:
1966 \l_@@_code_before_tl
1967 \q_stop
1968 }
```

```
1969 \keys_define:nn { nicematrix / CodeBefore }
1970 {
1971   create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1972   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1973   sub-matrix .value_required:n = true ,
1974   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1975   delimiters / color .value_required:n = true ,
1976   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1977 }

1978 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1979 {
1980   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1981   \@@_CodeBefore:w
1982 }
```

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.

```
1983 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1984 {
1985   \bool_if:NTF \g_@@_aux_found_bool
1986   {
1987     \@@_pre_code_before:
1988     \legacy_if:nF { measuring@ } { #1 }
1989   }
```

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).

```
1990 {
1991   \pgfsys@markposition { \@@_env: - position }
1992   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1993   \pgfpicture
1994   \pgf@relevantforpicturesizefalse
1995   \endpgfpicture
```

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

```
1996 \pgfpicture
1997 \pgfrememberpicturepositiononpagetrue
1998 \endpgfpicture
```

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

```
1999 \pgfpicture
2000 \pgf@relevantforpicturesizefalse
```

```

2001         \pgfrememberpicturerepositiononpagetrue
2002     \endpgfpicture

```

The following picture corresponds \@@_actually_color:

```

2003     \pgfpicture
2004     \pgf@relevantforpicturesizefalse
2005     \endpgfpicture
2006 }
2007 }

```

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.

```

2008 \cs_new_protected:Npn \@@_recreate_cell_nodes:
2009 {
2010     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
2011     {
2012         \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
2013         \pgfcoordinate { \@@_env: - row - ##1 - base }
2014         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
2015         \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
2016         {
2017             \cs_if_exist:cT
2018             { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
2019             {
2020                 \pgfsys@getposition
2021                 { \@@_env: - ##1 - #####1 - NW }
2022                 \@@_node_position:
2023                 \pgfsys@getposition
2024                 { \@@_env: - ##1 - #####1 - SE }
2025                 \@@_node_position_i:
2026                 \@@_pgf_rect_node:nnn
2027                 { \@@_env: - ##1 - #####1 }
2028                 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
2029                 { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
2030             }
2031         }
2032     }
2033     \@@_create_extra_nodes:
2034     \@@_create_aliases_last:
2035 }

```

```

2036 \cs_new_protected:Npn \@@_create_aliases_last:
2037 {
2038     \int_step_inline:nn \c@iRow
2039     {
2040         \pgfnodealias
2041         { \@@_env: - ##1 - last }
2042         { \@@_env: - ##1 - \int_use:N \c@jCol }
2043     }
2044     \int_step_inline:nn \c@jCol
2045     {
2046         \pgfnodealias
2047         { \@@_env: - last - ##1 }
2048         { \@@_env: - \int_use:N \c@iRow - ##1 }
2049     }
2050     \pgfnodealias
2051     { \@@_env: - last - last }
2052     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
2053 }

```

```

2054 \cs_new_protected:Npn \@@_create_blocks_nodes:

```

```

2055 {
2056   \pgfpicture
2057   \pgf@relevantforpicturesizefalse
2058   \pgfrememberpicturepositiononpagetrue
2059   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
2060   { \@@_create_one_block_node:nnnnn ##1 }
2061   \endpgfpicture
2062 }

```

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.⁸

```

2063 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
2064 {
2065   \tl_if_empty:nF { #5 }
2066   {
2067     \@@_qpoint:n { col - #2 }
2068     \dim_set_eq:NN \l_tmpa_dim \pgf@x
2069     \@@_qpoint:n { #1 }
2070     \dim_set_eq:NN \l_tmpb_dim \pgf@y
2071     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
2072     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
2073     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
2074     \@@_pgf_rect_node:nnnnn
2075     { \@@_env: - #5 }
2076     { \dim_use:N \l_tmpa_dim }
2077     { \dim_use:N \l_tmpb_dim }
2078     { \dim_use:N \l_@@_tmpc_dim }
2079     { \dim_use:N \pgf@y }
2080   }
2081 }

```

10 The environment `{NiceArrayWithDelims}`

```

2082 \NewDocumentEnvironment { NiceArrayWithDelims }
2083 { m m 0 { } m ! 0 { } t \CodeBefore }
2084 {
2085   \@@_provide_pgfsyspdfmark:
2086   \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.

```

2087   \bgroup

2088   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2089   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2090   \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
2091   \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

2092   \int_gzero:N \g_@@_block_box_int
2093   \dim_gzero:N \g_@@_width_last_col_dim
2094   \dim_gzero:N \g_@@_width_first_col_dim
2095   \bool_gset_false:N \g_@@_row_of_col_done_bool
2096   \str_if_empty:NT \g_@@_name_env_str
2097   { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
2098   \bool_if:NTF \l_@@_tabular_bool
2099     \mode_leave_vertical:
2100     \@@_test_if_math_mode:

```

⁸Moreover, 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).

```

2101 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
2102 \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁹. 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.

```

2103 \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.

```

2104 \cs_if_exist:NT \tikz@library@external@loaded
2105 {
2106   \tikzexternaldisable
2107   \cs_if_exist:NT \ifstandalone
2108     { \tikzset { external / optimize = false } }
2109 }

```

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

```

2110 \int_gincr:N \g_@@_env_int
2111 \bool_if:NF \l_@@_block_auto_columns_width_bool
2112 { \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.

```

2113 \seq_gclear:N \g_@@_blocks_seq
2114 \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`.

```

2115 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
2116 \seq_gclear:N \g_@@_pos_of_xdots_seq
2117 \tl_gclear_new:N \g_@@_code_before_tl
2118 \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.

```

2119 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2120 {
2121   \bool_gset_true:N \g_@@_aux_found_bool
2122   \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2123 }
2124 { \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.

```

2125 \tl_gclear:N \g_@@_aux_tl
2126 \tl_if_empty:NF \g_@@_code_before_tl
2127 {
2128   \bool_set_true:N \l_@@_code_before_bool
2129   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
2130 }
2131 \tl_if_empty:NF \g_@@_pre_code_before_tl
2132 { \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`.

```

2133 \bool_if:NTF \g_@@_delims_bool
2134 { \keys_set:nn { nicematrix / pNiceArray } }
2135 { \keys_set:nn { nicematrix / NiceArray } }

```

⁹e.g. `\color[rgb]{0.5,0.5,0}`

```
2136 { #3 , #5 }
```

```
2137 \@@_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:.

```
2138 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
2139 }
```

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

```
2140 {
2141   \bool_if:NTF \l_@@_light_syntax_bool
2142   { \use:c { end @@-light-syntax } }
2143   { \use:c { end @@-normal-syntax } }
2144   $ % $
2145   \skip_horizontal:N \l_@@_right_margin_dim
2146   \skip_horizontal:N \l_@@_extra_right_margin_dim
2147   \hbox_set_end:
2148   \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.

```
2149 \bool_if:NT \l_@@_width_used_bool
2150 {
2151   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2152   { \@@_error_or_warning:n { width~without~X~columns } }
2153 }
```

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

```
2154 \fp_compare:nNnT \g_@@_total_X_weight_fp > \c_zero_fp
2155 \@@_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).

```
2156 \int_compare:nNnT \l_@@_last_row_int > { -2 }
2157 {
2158   \bool_if:NF \l_@@_last_row_without_value_bool
2159   {
2160     \int_compare:nNnF \l_@@_last_row_int = \c_iRow
2161     {
2162       \@@_error_or_warning:n { Wrong~last~row }
2163       \int_set_eq:NN \l_@@_last_row_int \c_iRow
2164     }
2165   }
2166 }
```

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”.¹⁰

```
2167 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2168 \bool_if:NTF \g_@@_last_col_found_bool
2169 { \int_gdecr:N \c@jCol }
2170 { }
```

¹⁰We remind that the potential “first column” (exterior) has the number 0.

```

2171 \int_compare:nNtT \l_@@_last_col_int > { -1 }
2172 { \@@_error:n { last~col~not~used } }
2173 }

```

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

```

2174 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2175 \int_compare:nNtT \l_@@_last_row_int > { -1 }
2176 { \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. 98).

```

2177 \int_if_zero:nT \l_@@_first_col_int
2178 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2179 \bool_if:nTF { ! \g_@@_delims_bool }
2180 {
2181   \str_if_eq:eeTF c \l_@@_baseline_tl
2182   \@@_use_arraybox_with_notes_c:
2183   {
2184     \str_if_eq:eeTF b \l_@@_baseline_tl
2185     \@@_use_arraybox_with_notes_b:
2186     \@@_use_arraybox_with_notes:
2187   }
2188 }

```

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).

```

2189 {
2190   \int_if_zero:nTF \l_@@_first_row_int
2191   {
2192     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2193     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2194   }
2195   { \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”.¹¹

```

2196 \int_compare:nNtTF \l_@@_last_row_int > { -2 }
2197 {
2198   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2199   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2200 }
2201 { \dim_zero:N \l_tmpb_dim }
2202 \hbox_set:Nn \l_tmpa_box
2203 {
2204   \m@th
2205   $ % $
2206   \@@_color:o \l_@@_delimiters_color_tl
2207   \exp_after:wN \left \g_@@_left_delim_tl
2208   \vcenter
2209   {

```

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.

```

2210 \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2211 \hbox
2212 {
2213   % \bool_if:NtF \l_@@_tabular_bool
2214   % { \skip_horizontal:n { - \tabcolsep } }
2215   % { \skip_horizontal:n { - \arraycolsep } }

```

¹¹A 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).

```

2216         \skip_horizontal:n
2217         { - \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
2218     \@@_use_arraybox_with_notes_c:
2219     \skip_horizontal:n
2220     { - \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
2221     % \bool_if:NTF \l_@@_tabular_bool
2222     %   { \skip_horizontal:n { - \tabcolsep } }
2223     %   { \skip_horizontal:n { - \arraycolsep } }
2224 }

```

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

```

2225     \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2226 }
2227 \exp_after:wN \right \g_@@_right_delim_tl
2228 $ % $
2229 }

```

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.

```

2230     \bool_if:NTF \l_@@_delimiters_max_width_bool
2231     { \@@_put_box_in_flow_bis:nn \g_@@_left_delim_tl \g_@@_right_delim_tl }
2232     \@@_put_box_in_flow:
2233 }

```

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. 99).

```

2234     \bool_if:NT \g_@@_last_col_found_bool
2235     { \skip_horizontal:N \g_@@_width_last_col_dim }
2236     \@@_test_false_columns:
2237     \bool_if:NT \l_@@_preamble_bool
2238     {
2239         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2240         { \@@_err_columns_not_used: }
2241     }
2242     \@@_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.

```

2243     \egroup

```

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

```

2244     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2245     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2246     \iow_now:Ne \@mainaux
2247     {
2248         \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2249         \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2250         { \exp_not:o \g_@@_aux_tl }
2251     }
2252     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2253     \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2254 }

```

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

```

2255 \cs_new_protected:Npn \@@_err_columns_not_used:
2256 {
2257     \@@_warning:n { columns~not~used }
2258     \cs_gset:Npn \@@_err_columns_not_used: { }
2259 }

```

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 .

```

2260 \cs_new_protected:Npn \@@_compute_width_X:
2261 {
2262   \tl_gput_right:Ne \g_@@_aux_tl
2263   {
2264     \bool_set_true:N \l_@@_X_columns_aux_bool
2265     \dim_set:Nn \l_@@_X_columns_dim
2266     {

```

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`

```

2267       \bool_lazy_and:nnTF \g_@@_V_of_X_bool \l_@@_X_columns_aux_bool
2268       { \dim_use:N \l_@@_X_columns_dim }
2269       {
2270         \dim_compare:nNnTF
2271         {
2272           \dim_abs:n
2273           { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2274         }
2275         <
2276         { 0.001 pt }
2277         { \dim_use:N \l_@@_X_columns_dim }
2278         {
2279           \dim_eval:n
2280           {
2281             \l_@@_X_columns_dim
2282             +
2283             \fp_to_dim:n
2284             {
2285               (
2286                 \dim_eval:n
2287                 { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2288               )
2289               / \fp_use:N \g_@@_total_X_weight_fp
2290             }
2291           }
2292         }
2293       }
2294     }
2295   }
2296 }

```

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 `\l_@@_array_preamble_tl`.

```

2297 \cs_new_protected:Npn \@@_transform_preamble:
2298 {

```

First, some tuning.

```

2299   \@@_transform_preamble_i:

```

We will now actually make the preamble. The preamble provided by the final user (when he uses an environment with preamble, such as `{NiceTabular}`, etc.) has been stored in

`\g_@@_user_preamble_tl`. Using that information, we will create `\l_@@_array_preamble_tl`, which will be the preamble that we will provide to `{array}`.

```
2300 \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
```

Now, some post-treatment.

```
2301 \@@_transform_preamble_ii:
2302 }
2303 \cs_new_protected:Npn \@@_transform_preamble_i:
2304 {
```

The following counter will be used during the construction of `\l_@@_array_preamble_tl`, and, after that construction, it will be the number of columns of the tabular as indicated by the user preamble (for the environments with preamble).

```
2305 \int_zero:N \g_@@_static_num_of_col_int
```

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

```
2306 \seq_gclear:N \g_@@_cols_vlism_seq
2307 \clist_gclear:N \g_@@_false_cols_clist
2308 \clist_gclear:N \g_@@_false_rows_clist
```

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

```
2309 \int_zero:N \l_tmpa_int
2310 \str_if_eq:eeTF \l_@@_vlines_clist { all }
2311 {
2312   \tl_set:Nn \l_@@_array_preamble_tl
2313     { ! { \skip_horizontal:N \arrayrulewidth } }
2314 }
2315 {
2316   \clist_if_in:NnT \l_@@_vlines_clist 1
2317   {
2318     \tl_set:Nn \l_@@_array_preamble_tl
2319       { ! { \skip_horizontal:N \arrayrulewidth } }
2320   }
2321 }
2322 }
```

```
2323 \cs_new_protected:Npn \@@_transform_preamble_ii:
2324 {
2325   \@@_replace_columncolor:
```

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

```
2326 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2327 {
2328   \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2329   { \bool_gset_true:N \g_@@_delims_bool }
2330 }
2331 { \bool_gset_true:N \g_@@_delims_bool }
```

We complete the preamble with the potential “exterior columns” (on both sides).

```
2332 \int_if_zero:nTF \l_@@_first_col_int
2333 { \tl_put_left:No \l_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2334 {
2335   \bool_if:NF \g_@@_delims_bool
2336   {
2337     \bool_if:NF \l_@@_tabular_bool
2338     {
2339       \clist_if_empty:NT \l_@@_vlines_clist
2340       {
```

```

2341         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2342         { \tl_put_left:Nn \l_@@_array_preamble_tl { @ { } } }
2343     }
2344 }
2345 }
2346 }
2347 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2348 { \tl_put_right:No \l_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2349 {
2350     \bool_if:NF \g_@@_delims_bool
2351     {
2352         \bool_if:NF \l_@@_tabular_bool
2353         {
2354             \clist_if_empty:NT \l_@@_vlines_clist
2355             {
2356                 \bool_if:NF \l_@@_exterior_arraycolsep_bool
2357                 { \tl_put_right:Nn \l_@@_array_preamble_tl { @ { } } }
2358             }
2359         }
2360     }
2361 }

```

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.

```

2362     \tag_if_active:F
2363     {

```

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

```

2364     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2365     {
2366         \tl_put_right:Nn \l_@@_array_preamble_tl
2367         { > { \@@_err_too_many_cols: } 1 }
2368     }
2369 }
2370 }

```

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.

```

2371 \cs_new_protected:Npn \@@_rec_preamble:n #1
2372 {

```

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.¹²

```

2373     \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2374     { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2375     {

```

Now, the columns defined by \newcolumntype of array.

```

2376     \cs_if_exist:cTF { NC @ find @ #1 }
2377     {
2378         \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2379         \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl

```

¹²We do that because it's an easy way to insert the letter at some places in the code that we will add to \l_@@_array_preamble_tl.

```

2380     }
2381     {
2382         \str_case:nnF { #1 }
2383         {
2384             { S } { \@@_fatal:n { unknown-column-type~S } }
2385             { ; } { \@@_fatal:n { unknown-column-type~; } }
2386             { R } { \@@_fatal:nn { unknown-column-type~RCL } { #1 } }
2387             { C } { \@@_fatal:nn { unknown-column-type~RCL } { #1 } }
2388             { L } { \@@_fatal:nn { unknown-column-type~RCL } { #1 } }
2389         }
2390         { \@@_fatal:nn { unknown-column-type } { #1 } }
2391     }
2392 }
2393 }

2394 \cs_new_protected:cpn { @@_> : } #1 #2
2395 {
2396     \tl_put_right:Nn \l_@@_pre_cell_tl { > { #2 } }
2397     \@@_rec_preamble:n
2398 }

2399 \cs_new_protected:Npn \@@_use_pre_cell_tl:
2400 {
2401     \tl_put_right:No \l_@@_array_preamble_tl \l_@@_pre_cell_tl
2402     \tl_clear:N \l_@@_pre_cell_tl
2403 }

```

For c, l and r

```

2404 \cs_new_protected:Npn \@@_c: #1
2405 {
2406     \@@_use_pre_cell_tl:
2407     \tl_put_right:Nn \l_@@_array_preamble_tl
2408     { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2409     \@@_rec_preamble_after_col:n
2410 }

2411 \cs_new_protected:Npn \@@_l: #1
2412 {
2413     \@@_use_pre_cell_tl:
2414     \tl_put_right:Nn \l_@@_array_preamble_tl
2415     {
2416         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2417         l
2418         < \@@_cell_end:
2419     }

```

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

```

2420     \@@_rec_preamble_after_col:n
2421 }

2422 \cs_new_protected:Npn \@@_r: #1
2423 {
2424     \@@_use_pre_cell_tl:
2425     \tl_put_right:Nn \l_@@_array_preamble_tl
2426     {
2427         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2428         r
2429         < \@@_cell_end:
2430     }

```

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

```

2431     \@@_rec_preamble_after_col:n
2432 }

```

For ! and @

```

2433 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2434 {
2435   \tl_put_right:Nn \l_@@_array_preamble_tl { #1 { #2 } }
2436   \@@_rec_preamble:n
2437 }
2438 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2439 \cs_new_protected:cpn { @@ _ | : } #1
2440 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2441   \int_incr:N \l_tmpa_int
2442   \@@_make_preamble_i_i:n
2443 }
2444 \cs_new_protected:Npn \@@_F: #1
2445 {
2446   \tl_put_right:Nn \l_@@_array_preamble_tl
2447   {
2448     > {
2449       \skip_horizontal:n { -2 \col@sep + \l_@@_width_of_false_dim }
2450       \skip_horizontal:N \c_zero_dim
2451       \@@_cell_begin:
2452     }
2453     c
2454     < {
2455       \@@_cell_end:
2456     }
2457   }

```

The clist \g_@@_false_cols_clist is the list of the occurrences of the letter F (for the “false columns”).

```

2458   \clist_gput_right:Ne \g_@@_false_cols_clist
2459   { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
2460   \socket_assign_plug:nn { nicematrix / horizontal-space } { with-false-col }

```

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

```

2461   \@@_rec_preamble_after_col:n
2462 }
2463 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2464 {

```

Here, we can’t use \str_if_eq:eeTF.

```

2465   \str_if_eq:nnTF { #1 } { | }
2466   { \use:c { @@ _ | : } | }
2467   { \@@_make_preamble_i_ii:nn { } #1 }
2468 }

```

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

```

2469 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2470 {
2471   \str_if_eq:nnTF { #2 } { [ ] }
2472   { \@@_make_preamble_i_ii:nw { #1 } [ ] }
2473   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2474 }
2475 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2476 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2477 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2478 {
2479   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2480   \tl_put_right:Ne \l_@@_array_preamble_tl
2481   {

```

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

```

2482     \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_before_dim }
2483   }
2484   \tl_gput_right:Ne \g_@@_rules_tl
2485   {

```

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

```

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

```

We will use a version slightly more efficient:

```

2486   {
2487     \int_compare:nNnT \l_tmpa_int > 1
2488       { \@@_set_multiplicity:n { \int_use:N \l_tmpa_int } }
2489     \int_set:Nn \l_@@_position_int
2490       { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
2491     \dim_set:Nn \l_@@_rule_width_after_dim
2492       { \dim_use:N \l_@@_rule_width_before_dim }
2493     \@@_draw_vrule:n { #2 }
2494   }

```

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.

```

2495   }
2496   \int_zero:N \l_tmpa_int
2497   \str_if_eq:nnT { #1 } { \s_stop }
2498     { \bool_set_true:N \l_@@_bar_at_end_of_pream_bool }
2499   \@@_rec_preamble:n #1
2500 }

```

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.

```

2501 \keys_define:nn { nicematrix / p-column }
2502 {
2503   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2504   r .value_forbidden:n = true ,
2505   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2506   c .value_forbidden:n = true ,
2507   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2508   l .value_forbidden:n = true ,
2509   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2510   S .value_forbidden:n = true ,
2511   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2512   p .value_forbidden:n = true ,
2513   t .meta:n = p ,
2514   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2515   m .value_forbidden:n = true ,
2516   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2517   b .value_forbidden:n = true
2518 }

```

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

```

2519 \cs_new_protected:Npn \@@_p: #1
2520 {
2521   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2522 \@@_make_preamble_ii_i:n
2523 }
2524 \cs_new_eq:NN \@@_b: \@@_p:
2525 \cs_new_eq:NN \@@_m: \@@_p:
2526 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2527 {
2528   \str_if_eq:nnTF { #1 } { [ ] }
2529   { \@@_make_preamble_ii_ii:w [ ] }
2530   { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2531 }
2532 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2533 { \@@_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.

```

2534 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2535 {

```

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).

```

2536   \str_set:Nn \l_@@_hpos_col_str { j }
2537   \@@_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`.

```

2538   \setlength { \l_tmpa_dim } { #2 }
2539   \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2540 }
2541 \cs_new_protected:Npn \@@_keys_p_column:n #1
2542 { \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.

```

2543 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2544 {

```

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

```

2545   \use:e
2546   {
2547     \@@_make_preamble_ii_vi:nnnnnnnn
2548     { \str_if_eq:eeTF p \l_@@_vpos_col_str { t } { b } }
2549     { #1 }
2550     {
2551       \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.

```

2552       \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2553       { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2554       {

```

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

```

2555         \def \exp_not:N \l_@@_hpos_cell_tl
2556         { \str_lowercase:f { \l_@@_hpos_col_str } }
2557     }
2558     \IfPackageLoadedTF { ragged2e }
2559     {
2560         \str_case:on \l_@@_hpos_col_str
2561         {

```

The following `\exp_not:N` are mandatory.

```

2562         c { \exp_not:N \Centering }
2563         l { \exp_not:N \RaggedRight }
2564         r { \exp_not:N \RaggedLeft }
2565     }
2566 }
2567 {
2568     \str_case:on \l_@@_hpos_col_str
2569     {
2570         c { \exp_not:N \centering }
2571         l { \exp_not:N \raggedright }
2572         r { \exp_not:N \raggedleft }
2573     }
2574 }
2575 #3
2576 }
2577 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2578 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2579 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2580 { #2 }
2581 {
2582     \str_case:onF \l_@@_hpos_col_str
2583     {
2584         { j } { c }
2585         { si } { c }
2586     }

```

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

```

2587         { \str_lowercase:f \l_@@_hpos_col_str }
2588     }
2589 }

```

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

```

2590     \@@_rec_preamble_after_col:n
2591 }

```

#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*.

```

2592 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2593 {
2594     % \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2595     % {
2596     %     \tl_put_right:Nn \l_@@_array_preamble_tl

```

```

2597 % { > \@@_test_if_empty_for_S: }
2598 % }
2599 % {
2600 % \str_if_eq:eeTF { #7 } { varwidth }
2601 % {
2602 % \tl_put_right:Nn \l_@@_array_preamble_tl
2603 % { > \@@_test_if_empty_varwidth: }
2604 % }
2605 % { \tl_put_right:Nn \l_@@_array_preamble_tl { > \@@_test_if_empty: } }
2606 % }
2607 \tl_put_right:Ne \l_@@_array_preamble_tl
2608 {
2609 >
2610 \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2611 { \@@_test_if_empty_for_S: }
2612 {
2613 \str_if_eq:eeTF { #7 } { varwidth }
2614 \@@_test_if_empty_varwidth:
2615 \@@_test_if_empty:
2616 }
2617 }
2618 \@@_use_pre_cell_tl:
2619 \tl_put_right:Nn \l_@@_array_preamble_tl
2620 {
2621 > {

```

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.

```

2622 \dim_set:Nn \l_@@_col_width_dim { #2 }
2623 \@@_cell_begin:

```

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

```

2624 \use:c { #7 } [ #1 ] { #2 }

```

The following lines have been taken from `array.sty` (version 2.7b).

```

2625 \everypar
2626 {
2627 \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2628 \everypar { }
2629 }

```

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

```

2630 #3

```

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

```

2631 \g_@@_row_style_tl
2632 \arraybackslash
2633 #5
2634 }
2635 #8
2636 < {
2637 #6

```

The following line has been taken from `array.sty` (version 2.7b).

```

2638 \@finalstrut \@arstrutbox
2639 \use:c { end #7 }

```

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

```

2640 #4
2641 \@@_cell_end:
2642 }
2643 }
2644 }

```

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

```
2645 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2646 {
```

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 `&`).

```
2647 \group_align_safe_begin:
2648 \peek_meaning:NTF &
2649 \@@_the_cell_is_empty: % contains \group_align_safe_end:
2650 {
2651 \peek_meaning:NTF \\\
2652 \@@_the_cell_is_empty: % idem
2653 {
2654 \peek_meaning:NTF \crcr
2655 \@@_the_cell_is_empty: % idem
2656 \group_align_safe_end:
2657 }
2658 }
2659 }
```

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

```
2660 \cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
2661 {
2662 \group_align_safe_begin:
2663 \peek_meaning:NTF &
2664 \@@_the_cell_is_empty_varwidth:
2665 {
2666 \peek_meaning:NTF \\\
2667 \@@_the_cell_is_empty_varwidth:
2668 {
2669 \peek_meaning:NTF \crcr
2670 \@@_the_cell_is_empty_varwidth:
2671 \group_align_safe_end:
2672 }
2673 }
2674 }

2675 \cs_new_protected:Npn \@@_the_cell_is_empty:
2676 {
2677 \group_align_safe_end:
2678 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2679 {
```

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

```
2680 \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).

```
2681 \skip_horizontal:N \l_@@_col_width_dim
2682 }
2683 }

2684 \cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
2685 {
2686 \group_align_safe_end:
2687 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2688 { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2689 }
```

```

2690 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2691 {
2692   \peek_meaning:NT \__siunitx_table_skip:n
2693   { \bool_gset_true:N \g_@@_empty_cell_bool }
2694 }

```

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.

```

2695 \cs_new_protected:Npn \@@_center_cell_box:
2696 {

```

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

```

2697   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2698   {
2699     \dim_compare:nNnT
2700     { \box_ht:N \l_@@_cell_box }
2701     >

```

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).

```

2702     { \box_ht:N \strutbox }
2703     {
2704       \hbox_set:Nn \l_@@_cell_box
2705       {
2706         \box_move_down:nn
2707         {
2708           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2709             + \baselineskip ) / 2
2710         }
2711         { \box_use:N \l_@@_cell_box }
2712       }
2713     }
2714   }
2715 }

```

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

```

2716 \cs_new_protected:Npn \@@_V: #1 #2
2717 {
2718   \str_if_eq:nnTF { #2 } { [ ] }
2719   { \@@_make_preamble_V_i:w [ ] }
2720   { \@@_make_preamble_V_i:w [ ] { #2 } }
2721 }
2722 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2723 { \@@_make_preamble_V_ii:nn { #1 } }
2724 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2725 {
2726   \str_set:Nn \l_@@_vpos_col_str { p }
2727   \str_set:Nn \l_@@_hpos_col_str { j }
2728   \@@_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`.

```

2729   \setlength { \l_tmpa_dim } { #2 }
2730   \IfPackageLoadedTF { varwidth }
2731   { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2732   {

```

```

2733     \@@_error_or_warning:n { varwidth-not-loaded }
2734     \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2735   }
2736 }
2737 % \end{macrocod}
2738 %
2739 % \medskip
2740 % For |w| and |W|
2741 % \begin{macrocode}
2742 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2743 \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.

```

2744 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3
2745 {
2746   \tl_if_in:nnTF { clr } { #3 }
2747   { \@@_make_preamble_w_i:nnnn { #1 } { #2 } { #3 } }
2748   {
2749     \@@_error:nn { Invalid-argument-for-w } { #3 }
2750     \@@_make_preamble_w_i:nnnn { #1 } { #2 } { c }
2751   }
2752 }
2753 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2 #3 #4
2754 {
2755   \str_if_eq:nnTF { #3 } { s }
2756   { \@@_make_preamble_w_ii:nnnn { #1 } { #4 } }
2757   { \@@_make_preamble_w_iii:nnnn { #1 } { #2 } { #3 } { #4 } }
2758 }

```

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.

```

2759 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2
2760 {
2761   \@@_use_pre_cell_tl:
2762   \tl_put_right:Nn \l_@@_array_preamble_tl
2763   {
2764     > {

```

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.

```

2765     \setlength { \l_@@_col_width_dim } { #2 }
2766     \@@_cell_begin:
2767     \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2768   }
2769   c
2770   < {
2771     \@@_cell_end_for_w_s:
2772     #1
2773     \@@_adjust_size_box:
2774     % \box_use_drop:N \l_@@_cell_box
2775   }
2776 }

```

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

```

2777   \@@_rec_preamble_after_col:n
2778 }

```

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

```
2779 \cs_new_protected:Npn \@@_make_preamble_w_iii:nnnn #1 #2 #3 #4
2780 {
2781   \@@_use_pre_cell_tl:
```

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`.

We don't want to compute the width of the column with `\setlength` in each cell of the column. That's why we compute it now and use an expansion after.

We could do the same thing previously for a column `w` of type `s` but the type `s` is almost never used...

```
2782   \setlength { \l_tmpa_dim } { #4 }
2783   \tl_put_right:Ne \l_@@_array_preamble_tl
2784   {
2785     > {
```

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 need `\dim_use:N` because we are in an expansion.

```
2786       \dim_set:Nn \l_@@_col_width_dim { \dim_use:N \l_tmpa_dim }
2787       \hbox_set:Nw \l_@@_cell_box
2788       \@@_cell_begin:
2789       \tl_set:Nn \exp_not:N \l_@@_hpos_cell_tl { #3 }
2790     }
2791   }
2792   \tl_put_right:Nn \l_@@_array_preamble_tl
2793   {
2794     c
2795     < {
2796       \@@_cell_end:
2797       \hbox_set_end:
2798       #1
2799       \@@_adjust_size_box:
2800       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2801     }
2802   }
```

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

```
2803   \@@_rec_preamble_after_col:n
2804 }

2805 \cs_new_protected:Npn \@@_special_W:
2806 {
2807   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2808   { \@@_warning:n { W-warning } }
2809 }
```

For `S` (of `siunitx`).

```
2810 \AddToHook { package / siunitx / after }
2811 {
2812   \cs_new_protected:Npn \@@_S: #1 #2
2813   {
2814     \str_if_eq:nnTF { #2 } { [ ] }
2815     { \@@_make_preamble_S:w [ ] }
2816     { \@@_make_preamble_S:w [ ] { #2 } }
2817   }
2818 }

2819 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2820 { \@@_make_preamble_S:i:n { #1 } }
```

```

2821 \cs_new_protected:Npn \@@_test_siunitx:
2822 {
2823   \IfPackageAtLeastF { siunitx } { 2026/03/26 }
2824   { \@@_fatal:n { siunitx~too~old } }
2825   \cs_gset_eq:NN \@@_test_siunitx: \prg_do_nothing:
2826 }
2827 % \end{macrocode}
2828 %
2829 % \begin{macrocode}
2830 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2831 {
2832   \@@_test_siunitx:
2833   \@@_use_pre_cell_tl:
2834   \tl_put_right:Nn \l_@@_array_preamble_tl
2835   {
2836     > {

```

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).

```

2837       \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2838       \keys_set:nn { siunitx } { #1 }
2839       \@@_cell_begin:
2840       \siunitx_cell_begin:w
2841     }
2842     c
2843     <
2844     {
2845       \siunitx_cell_end:

```

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

```

2846       \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2847       {
2848         \bool_if:NTF \l__siunitx_table_text_bool
2849         \bool_set_true:N
2850         \bool_set_false:N
2851         \l__siunitx_table_text_bool
2852       }
2853       \@@_cell_end:
2854     }
2855   }

```

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

```

2856       \@@_rec_preamble_after_col:n
2857     }

```

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

```

2858 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2859 {
2860   \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.

```

2861   \int_if_zero:nTF \g_@@_static_num_of_col_int
2862   {
2863     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2864     {

```

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

```

2865       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2866       \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl

```

```

2867         \@@_rec_preamble:n #2
2868     }
2869     {
2870         \tl_put_right:Nn \l_@@_array_preamble_tl { ! { \enskip } }
2871         \@@_make_preamble_iv:nn { #1 } { #2 }
2872     }
2873 }
2874 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2875 }
2876 \cs_set_eq:cc { @@ _ \token_to_str:N [ : ] { @@ _ \token_to_str:N ( : }
2877 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2878 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2879 {
2880     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2881     {
2882         \@@_delimiter:nnn #1
2883         { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
2884         \c_true_bool
2885     }
2886     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2887     {
2888         \@@_error:nn { delimiter~after~opening } { #2 }
2889         \@@_rec_preamble:n
2890     }
2891     { \@@_rec_preamble:n #2 }
2892 }

```

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

```

2893 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2894 { \use:c { @@ _ \token_to_str:N ( : } }

```

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

```

2895 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2896 {
2897     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2898     \tl_if_in:nnTF { ) ] \} } { #2 }
2899     { \@@_make_preamble_v:nnn #1 #2 }
2900     {
2901         \str_if_eq:nnTF \s_stop { #2 }
2902         {
2903             \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2904             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2905             {
2906                 \tl_put_right:Nn \l_@@_array_preamble_tl { ! { \enskip } }
2907                 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2908                 {
2909                     \@@_delimiter:nnn #1
2910                     { \int_use:N \g_@@_static_num_of_col_int }
2911                     \c_false_bool
2912                 }
2913                 \@@_rec_preamble:n #2
2914             }
2915         }
2916         {
2917             \tl_if_in:nnT { ( [ \{ \left } { #2 }
2918             { \tl_put_right:Nn \l_@@_array_preamble_tl { ! { \enskip } } }
2919             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2920             {
2921                 \@@_delimiter:nnn #1
2922                 { \int_use:N \g_@@_static_num_of_col_int }

```

```

2923         \c_false_bool
2924     }
2925     \@@_rec_preamble:n #2
2926 }
2927 }
2928 }
2929 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2930 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2931 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2932 {
2933     \str_if_eq:nnTF \s_stop { #3 }
2934     {
2935         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2936         {
2937             \tl_put_right:Nn \l_@@_array_preamble_tl { ! { \enskip } }
2938             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2939             {
2940                 \@@_delimiter:nnn #1
2941                 { \int_use:N \g_@@_static_num_of_col_int }
2942                 \c_false_bool
2943             }
2944             \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2945         }
2946         {
2947             \tl_put_right:Nn \l_@@_array_preamble_tl { ! { \enskip } }
2948             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2949             {
2950                 \@@_delimiter:nnn #1
2951                 { \int_use:N \g_@@_static_num_of_col_int }
2952                 \c_false_bool
2953             }
2954             \@@_error:nn { double~closing~delimiter } { #2 }
2955         }
2956     }
2957     {
2958         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2959         {
2960             \@@_delimiter:nnn #1
2961             { \int_use:N \g_@@_static_num_of_col_int }
2962             \c_false_bool
2963         }
2964         \@@_error:nn { double~closing~delimiter } { #2 }
2965         \@@_rec_preamble:n #3
2966     }
2967 }

2968 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2969 { \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 @{...}.

```

2970 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2971 {
2972     % \int_gincr:N \g_@@_static_num_of_col_int
2973     \str_if_eq:nnTF { #1 } { < }
2974     { \@@_rec_preamble_after_col_i:n }
2975     {
2976         \int_gincr:N \g_@@_static_num_of_col_int % modified 2026-09-09
2977         \str_if_eq:nnTF { #1 } { @ }
2978         { \@@_rec_preamble_after_col_ii:n }
2979         {

```

```

2980     \str_if_eq:eeTF \l_@@_vlines_clist { all }
2981     {
2982         \tl_put_right:Nn \l_@@_array_preamble_tl
2983         { ! { \skip_horizontal:N \arrayrulewidth } }
2984     }
2985     {
2986         \clist_if_in:NeT \l_@@_vlines_clist
2987         { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
2988         {
2989             \tl_put_right:Nn \l_@@_array_preamble_tl
2990             { ! { \skip_horizontal:N \arrayrulewidth } }
2991         }
2992     }
2993     \@@_rec_preamble:n { #1 }
2994 }
2995 }
2996 }
2997 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2998 {
2999     \tl_put_right:Nn \l_@@_array_preamble_tl { < { #1 } }
3000     \@@_rec_preamble_after_col:n
3001 }

```

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.

```

3002 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
3003 {
3004     \str_if_eq:eeTF \l_@@_vlines_clist { all }
3005     {
3006         \tl_put_right:Nn \l_@@_array_preamble_tl
3007         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
3008     }
3009     {
3010         \clist_if_in:NeTF \l_@@_vlines_clist
3011         { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
3012         {
3013             \tl_put_right:Nn \l_@@_array_preamble_tl
3014             { @ { #1 \skip_horizontal:N \arrayrulewidth } }
3015         }
3016         { \tl_put_right:Nn \l_@@_array_preamble_tl { @ { #1 } } }
3017     }
3018     \@@_rec_preamble:n
3019 }

```

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.

```

3020 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
3021 { \@@_rec_preamble:n }

3022 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
3023 {
3024     \tl_clear:N \l_tmpa_tl

```

We use the neutral token \NC@find as a kind of \relax to stop the analysis after each iteration of the pattern.

```

3025     \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 \NC@find } }
3026     \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
3027 }

```

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.

```

3028 \cs_new_protected:Npn \@@_X: #1 #2
3029 {
3030   \str_if_eq:nnTF { #2 } { [ ] }
3031   { \@@_make_preamble_X:w [ ] }
3032   { \@@_make_preamble_X:w [ ] #2 }
3033 }
3034 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
3035 { \@@_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`.

```

3036 \keys_define:nn { nicematrix / X-column }
3037 {
3038   V .code:n =
3039   \IfPackageLoadedTF { varwidth }
3040   {
3041     \bool_set_true:N \l_@@_V_of_X_bool
3042     \bool_gset_true:N \g_@@_V_of_X_bool
3043   }
3044   { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
3045   unknown .code:n =
3046   \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
3047   { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
3048   { \@@_error_or_warning:n { invalid~weight } }
3049 }

```

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

```

3050 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
3051 {

```

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).

```

3052   \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).

```

3053   \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`).

```

3054   \fp_set:Nn \l_tmpa_fp { 1.0 }
3055   \@@_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.

```

3056   \bool_set_false:N \l_@@_V_of_X_bool
3057   \keys_set:no { nicematrix / X-column } \l_tmpa_tl

```

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

```

3058   \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).

```

3059   \bool_if:NTF \l_@@_X_columns_aux_bool
3060   {
3061     \@@_make_preamble_ii_iv:nnn

```

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

```

3062         { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
3063         { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
3064         { \@@_no_update_width: }
3065     }

```

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.

```

3066     {
3067         \tl_put_right:Nn \l_@@_array_preamble_tl
3068         {
3069             > {
3070                 \@@_cell_begin:
3071                 \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`.

```

3072         \NotEmpty

```

The following code will nullify the box of the cell.

```

3073         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
3074         { \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`.

```

3075         \begin { minipage } { 5 cm } \arraybackslash
3076     }
3077     c
3078     < {
3079         \end { minipage }
3080         \@@_cell_end:
3081     }
3082 }
3083 \@@_rec_preamble_after_col:n
3084 }
3085 }

```

```

3086 \cs_new_protected:Npn \@@_no_update_width:
3087 {
3088     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
3089     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
3090 }

```

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

```

3091 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
3092 {
3093     \seq_gput_right:Ne \g_@@_cols_vlism_seq
3094     { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
3095     \tl_put_right:Ne \l_@@_array_preamble_tl
3096     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
3097     \@@_rec_preamble:n
3098 }

```

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.

```

3099 \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).

```

3100 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
3101 { \@@_fatal:n { Preamble~forgotten } }
3102 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
3103 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
3104 { @@ _ \token_to_str:N \hline : }
3105 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
3106 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
3107 { @@ _ \token_to_str:N \hline : }
3108 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
3109 { @@ _ \token_to_str:N \hline : }
3110 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
3111 { @@ _ \token_to_str:N \hline : }
3112 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
3113 { @@ _ \token_to_str:N \hline : }
3114 \cs_new_protected:cpn { @@ _ \token_to_str:N \linewidth : }
3115 { \@@_fatal:n { NiceTabularX~probably~required } }
3116 \cs_set_eq:cc { @@ _ \token_to_str:N \textwidth : }
3117 { @@ _ \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).

```

3118 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
3119 {

```

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`.

```

3120 \multispan { #1 }
3121 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
3122 \begingroup
3123 \tbl_update_multicolumn_cell_data:n { #1 }

```

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

```

3124 \tl_gclear:N \g_@@_preamble_tl
3125 \@@_make_m_preamble:n #2 \q_stop

```

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

```

3126 \def \@addamp
3127 {
3128 \legacy_if:nTF { @firstamp }
3129 { \legacy_if_set_false:n { @firstamp } }
3130 { \@preamerr 5 }
3131 }
3132 \exp_args:No \mkpream \g_@@_preamble_tl
3133 \@addtopreamble \empty
3134 \endgroup
3135 \UseTaggingSocket { tbl / colspan } { #1 }

```

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

```

3136 \int_compare:nNnT { #1 } > 1
3137 {
3138 \seq_gput_right:Ne \g_@@_multicolumn_cells_seq
3139 { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
3140 \seq_gput_right:Nn \g_@@_multicolumn_sizes_seq { #1 }

```

```

3141 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
3142 {
3143   {
3144     \int_if_zero:nTF \c@jCol
3145       { \int_eval:n { \c@iRow + 1 } }
3146       { \int_use:N \c@iRow }
3147   }
3148   { \int_eval:n { \c@jCol + 1 } }
3149   {
3150     \int_if_zero:nTF \c@jCol
3151       { \int_eval:n { \c@iRow + 1 } }
3152       { \int_use:N \c@iRow }
3153   }
3154   { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block.

```

3155   { }
3156 }
3157 }

```

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

```

3158 \RenewDocumentCommand { \cellcolor } { 0 { } m }
3159 {
3160   \tl_gput_right:Ne \g_@@_pre_code_before_tl
3161   {
3162     \@@_rectanglecolor [ ##1 ]
3163     { \exp_not:n { ##2 } }
3164     { \int_use:N \c@iRow - \int_use:N \c@jCol }
3165     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
3166   }
3167   \ignorespaces
3168 }

```

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

```

3169 \def \@sharp { #3 }
3170 \@arstrut
3171 \@preamble
3172 \null

```

We add some lines.

```

3173 \int_gadd:Nn \c@jCol { #1 - 1 }
3174 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
3175   { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3176 \ignorespaces
3177 }

```

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`.

```

3178 \
3179 \cs_new_protected:Npn \@_make_m_preamble:n #1
3180 {
3181   \str_case:nnF { #1 }
3182   {
3183     c { \@_make_m_preamble_i:n #1 }
3184     l { \@_make_m_preamble_i:n #1 }
3185     X { \@_make_m_preamble_i:n l }
3186     r { \@_make_m_preamble_i:n #1 }
3187     > { \@_make_m_preamble_ii:nn #1 }
3188     ! { \@_make_m_preamble_ii:nn #1 }
3189     @ { \@_make_m_preamble_ii:nn #1 }
3190     | { \@_make_m_preamble_iii:n #1 }

```

```

3191 p { \@@_make_m_preamble_iv:nnn t #1 }
3192 m { \@@_make_m_preamble_iv:nnn c #1 }
3193 b { \@@_make_m_preamble_iv:nnn b #1 }
3194 w { \@@_make_m_preamble_v:nnnn { } #1 }
3195 W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
3196 \q_stop { }
3197 }
3198 {
3199 \cs_if_exist:cTF { NC @ find @ #1 }
3200 {
3201 \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
3202 \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
3203 }
3204 {
3205 \str_if_eq:nnTF { #1 } { S }
3206 { \@@_fatal:n { unknown~column~type~S~multicolumn } }
3207 { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
3208 }
3209 }
3210 }

```

For c, l and r

```

3211 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
3212 {
3213 \tl_gput_right:Nn \g_@@_preamble_tl
3214 {
3215 > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
3216 #1
3217 < \@@_cell_end:
3218 }

```

We test for the presence of a <.

```

3219 \@@_make_m_preamble_x:n
3220 }

```

For >, ! and @

```

3221 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
3222 {
3223 \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
3224 \@@_make_m_preamble:n
3225 }

```

For |

```

3226 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
3227 {
3228 \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
3229 \@@_make_m_preamble:n
3230 }

```

For p, m and b

```

3231 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
3232 {
3233 \tl_gput_right:Nn \g_@@_preamble_tl
3234 {
3235 > {
3236 \@@_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`.

```

3237 \setlength { \l_tmpa_dim } { #3 }
3238 \begin { minipage } [ #1 ] { \l_tmpa_dim }
3239 \mode_leave_vertical:

```

```

3240         \arraybackslash
3241         \vrule height \box_ht:N \@arstrutbox depth \c_zero_dim width \c_zero_dim
3242     }
3243     c
3244     < {
3245         \vrule height \c_zero_dim depth \box_dp:N \@arstrutbox width \c_zero_dim
3246         \end { minipage }
3247         \@@_cell_end:
3248     }
3249 }

```

We test for the presence of a <.

```

3250     \@@_make_m_preamble_x:n
3251 }

```

For w and W

```

3252 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3253 {
3254     \tl_gput_right:Nn \g_@@_preamble_tl
3255     {
3256         > {
3257             \dim_set:Nn \l_@@_col_width_dim { #4 }
3258             \hbox_set:Nw \l_@@_cell_box
3259             \@@_cell_begin:
3260             \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3261         }
3262         c
3263         < {
3264             \@@_cell_end:
3265             \hbox_set_end:
3266             \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3267             #1
3268             \@@_adjust_size_box:
3269             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3270         }
3271     }

```

We test for the presence of a <.

```

3272     \@@_make_m_preamble_x:n
3273 }

```

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

```

3274 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3275 {
3276     \str_if_eq:nnTF { #1 } { < }
3277     \@@_make_m_preamble_ix:n
3278     { \@@_make_m_preamble:n { #1 } }
3279 }
3280 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3281 {
3282     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3283     \@@_make_m_preamble_x:n
3284 }

```

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).

```

3285 \cs_new_protected:Npn \@@_put_box_in_flow:
3286 {
3287     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3288     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3289     \str_if_eq:eeTF \l_@@_baseline_tl { c }

```

```

3290 { \box_use_drop:N \l_tmpa_box }
3291 \@@_put_box_in_flow_i:
3292 }

```

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

```

3293 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3294 {
3295   \pgfpicture
3296     \@@_qpoint:n { row - 1 }
3297     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3298     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3299     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3300     \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).

```

3301   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3302   {
3303     \int_set:Nn \l_tmpa_int
3304       { \str_range:Nnn \l_@@_baseline_tl { 6 } { -1 } }
3305     \bool_lazy_or:nnT
3306       { \int_compare_p:nNn \l_tmpa_int < { 1 } }
3307       { \int_compare_p:nNn \l_tmpa_int > { \c@iRow + 1 } }
3308     {
3309       \@@_error:n { bad-value-for-baseline-line }
3310       \int_set:Nn \l_tmpa_int 1
3311     }
3312     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3313   }
3314   {
3315     \str_if_eq:eeTF t \l_@@_baseline_tl
3316     { \int_set:Nn \l_tmpa_int 1 }
3317     {
3318       \str_if_eq:eeTF b \l_@@_baseline_tl
3319       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3320       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3321     }
3322     \bool_lazy_or:nnT
3323       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3324       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3325     {
3326       \@@_error:n { bad-value-for-baseline }
3327       \int_set:Nn \l_tmpa_int 1
3328     }
3329     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3330     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3331   }
3332   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3333   \endpgfpicture
3334   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3335 }

```

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).

```

3336 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3337 {

```

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.

```

3338   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3339   {
3340     \int_compare:nNnT \c@jCol > 1
3341     {
3342       \box_set_wd:Nn \l_@@_the_array_box
3343       { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3344     }
3345   }

```

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

```

3346   \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3347   \bool_if:NT \l_@@_caption_above_bool
3348   {
3349     \tl_if_empty:NF \l_@@_caption_tl
3350     {
3351       \bool_set_false:N \g_@@_caption_finished_bool
3352       \int_gzero:N \c@tabularnote
3353       \@@_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).

```

3354       \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3355       {
3356         \tl_gput_right:Ne \g_@@_aux_tl
3357         {
3358           \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3359           { \int_use:N \g_@@_notes_caption_int }
3360         }
3361         \int_gzero:N \g_@@_notes_caption_int
3362       }
3363     }
3364   }

```

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

```

3365   \hbox
3366   {
3367     \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.

```

3368     \@@_create_extra_nodes:
3369     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3370   }

```

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).

```

3371   \bool_lazy_any:nT
3372   {
3373     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3374     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3375     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3376   }
3377   {
3378     \bool_if:NTF \l_@@_notes_no_print_bool
3379     { \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes: }

```

```

3380         \@@_tabular_notes:
3381     }
3382     \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3383     \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3384     \end { minipage }
3385 }

```

```

3386 \cs_new_protected:Npn \@@_insert_caption:
3387 {
3388     \tl_if_empty:NF \l_@@_caption_tl
3389     {
3390         \cs_if_exist:NTF \@capttype
3391         { \@@_insert_caption_i: }
3392         { \@@_error:n { caption~outside~float } }
3393     }
3394 }

```

```

3395 \cs_new_protected:Npn \@@_insert_caption_i:
3396 {
3397     \group_begin:

```

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

```

3398     \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.

```

3399     \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
3400     \tl_if_empty:NTF \l_@@_short_caption_tl
3401     \caption
3402     { \caption [ \l_@@_short_caption_tl ] }
3403     { \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.

```

3404     \bool_if:NF \g_@@_caption_finished_bool
3405     {
3406         \bool_gset_true:N \g_@@_caption_finished_bool
3407         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3408         \int_gzero:N \c@tabularnote
3409     }
3410     \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3411     \group_end:
3412 }

```

```

3413 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3414 {
3415     \@@_error_or_warning:n { tabularnote~below~the~tabular }
3416     \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3417 }

```

```

3418 \cs_set_protected:Npn \@@_tabular_notes_error:
3419 { \@@_error:n { Misuse~of~NiceTabularNotes } }

```

```

3420 \cs_set_eq:NN \NiceTabularNotes \@@_tabular_notes_error:

```

```

3421 \cs_set_protected:Npn \@@_tabular_notes:
3422 {
3423     \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes_error:
3424     \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3425     \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3426     \skip_vertical:N 0.65ex

```

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

```

3427 \group_begin:
3428 \l_@@_notes_code_before_tl
3429 \tl_if_empty:NF \g_@@_tabularnote_tl
3430 {
3431   \g_@@_tabularnote_tl \par
3432   \tl_gclear:N \g_@@_tabularnote_tl
3433 }

```

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.

```

3434 \int_compare:nNnT \c@tabularnote > \c_zero_int
3435 {
3436   \bool_if:NTF \l_@@_notes_para_bool
3437   {
3438     \begin { tabularnotes* }
3439     \seq_map_inline:Nn \g_@@_notes_seq { \@@_one_tabularnote:nn ##1 }
3440     \strut
3441     \end { tabularnotes* }

```

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

```

3442 \par
3443 }
3444 {
3445   \tabularnotes
3446   \seq_map_inline:Nn \g_@@_notes_seq { \@@_one_tabularnote:nn ##1 }
3447   \strut
3448   \endtabularnotes
3449 }
3450 }
3451 \unskip
3452 \group_end:
3453 \bool_if:NT \l_@@_notes_bottomrule_bool
3454 {
3455   \IfPackageLoadedTF { booktabs }
3456   {

```

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

```

3457 \skip_vertical:N \aboverulesep

```

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

```

3458 { \CT@arc@ \hrule height \heavyrulewidth }
3459 }
3460 { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3461 }
3462 \l_@@_notes_code_after_tl
3463 \seq_gclear:N \g_@@_notes_seq
3464 \seq_gclear:N \g_@@_notes_in_caption_seq
3465 \int_gzero:N \c@tabularnote
3466 }

```

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.

```

3467 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3468 {
3469   \tl_if_novalue:nTF { #1 }
3470   { \item }
3471   { \item [ \@@_notes_label_in_list:n { #1 } ] }
3472 }

```

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`.

```

3473 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3474 {
3475   \pgfpicture
3476     \@@_qpoint:n { row - 1 }
3477     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3478     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3479     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3480   \endpgfpicture
3481   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3482   \int_if_zero:nT \l_@@_first_row_int
3483   {
3484     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3485     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3486   }
3487   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3488 }

```

Now, the general case.

```

3489 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3490 {

```

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

```

3491   \str_if_eq:eeT \l_@@_baseline_tl { t }
3492   { \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`.

```

3493   \pgfpicture
3494   \@@_qpoint:n { row - 1 }
3495   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3496   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3497   {
3498     \int_set:Nn \l_tmpa_int
3499     {
3500       \str_range:Nnn
3501         \l_@@_baseline_tl
3502         { 6 }
3503         { \tl_count:o \l_@@_baseline_tl }
3504     }
3505     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3506   }
3507   {
3508     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3509     \bool_lazy_or:nnT
3510     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3511     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3512     {
3513       \@@_error:n { bad-value-for-baseline }
3514       \int_set:Nn \l_tmpa_int 1
3515     }
3516     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3517   }
3518   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3519   \endpgfpicture
3520   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3521   \int_if_zero:nT \l_@@_first_row_int
3522   {
3523     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3524     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3525   }
3526   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }

```

```
3527 }
```

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.

```
3528 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3529 {
```

We will compute the real width of both delimiters used.

```
3530 \dim_zero_new:N \l_@@_real_left_delim_dim
3531 \dim_zero_new:N \l_@@_real_right_delim_dim
3532 \hbox_set:Nn \l_tmpb_box
3533 {
3534   \m@th
3535   $ % $
3536   \left #1
3537   \vcenter
3538   {
3539     \vbox_to_ht:nn
3540     { \box_ht_plus_dp:N \l_tmpa_box }
3541     { }
3542   }
3543   \right .
3544   $ % $
3545 }
3546 \dim_set:Nn \l_@@_real_left_delim_dim
3547 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3548 \hbox_set:Nn \l_tmpb_box
3549 {
3550   \m@th
3551   $ % $
3552   \left .
3553   \vbox_to_ht:nn
3554   { \box_ht_plus_dp:N \l_tmpa_box }
3555   { }
3556   \right #2
3557   $ % $
3558 }
3559 \dim_set:Nn \l_@@_real_right_delim_dim
3560 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

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

```
3561 \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3562 \@@_put_box_in_flow:
3563 \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3564 }
```

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).

```
3565 \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).

```
3566 {
3567   \peek_remove_spaces:n
3568   {
3569     \peek_meaning:NTF \end
3570     \@@_analyze_end:Nn
```

```

3571     {
3572         \@@_transform_preamble:
3573         \@@_array:o \l_@@_array_preamble_tl
3574     }
3575 }
3576 }
3577 {
3578     \@@_create_col_nodes:
3579     \endarray
3580 }

```

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

```

3581 \NewDocumentEnvironment { @@-light-syntax } { b }
3582 {

```

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`.

```

3583     \tl_if_empty:nT { #1 }
3584     { \@@_fatal:n { empty-environment } }
3585     \tl_if_in:nnT { #1 } { & }
3586     { \@@_fatal:n { ampersand-in-light-syntax } }
3587     \tl_if_in:nnT { #1 } { \ }
3588     { \@@_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`.

```

3589     \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3590 }

```

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.

```

3591 {
3592     \@@_create_col_nodes:
3593     \endarray
3594 }
3595 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3596 {
3597     \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).

```

3598     \seq_clear_new:N \l_@@_rows_seq

```

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

```

3599     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3600     \bool_if:NTF \l_@@_light_syntax_expanded_bool
3601         \seq_set_split:Nee
3602         \seq_set_split:Non
3603         \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3604     \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3605     \tl_if_empty:NF \l_tmpa_tl
3606     { \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.

```

3607   \int_compare:nNnT \l_@@_last_row_int = { -1 }
3608   { \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`).

```

3609   \tl_build_begin:N \l_@@_new_body_tl
3610   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3611   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3612   \@@_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).

```

3613   \seq_map_inline:Nn \l_@@_rows_seq
3614   {
3615     \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3616     \@@_line_with_light_syntax:n { ##1 }
3617   }
3618   \tl_build_end:N \l_@@_new_body_tl
3619   \int_compare:nNnT \l_@@_last_col_int = { -1 }
3620   {
3621     \int_set:Nn \l_@@_last_col_int
3622     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3623   }

```

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.

```

3624   \@@_transform_preamble:

3625   \@@_array:o \l_@@_array_preamble_tl \l_@@_new_body_tl
3626   }

3627   \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3628   {
3629     \seq_clear_new:N \l_@@_cells_seq
3630     \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3631     \int_set:Nn \l_@@_nb_cols_int
3632     { \int_max:nn \l_@@_nb_cols_int { \seq_count:N \l_@@_cells_seq } }
3633     \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3634     \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3635     \seq_map_inline:Nn \l_@@_cells_seq
3636     { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3637   }
3638   \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`.

```

3639   \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3640   {
3641     \str_if_eq:eeT \g_@@_name_env_str { #2 }
3642     { \@@_fatal:n { empty-environment } }

```

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

```

3643   \end { #2 }
3644   }

```

The following command will be used in `\@@_create_col_nodes`.

```

3645 \socket_new:nn { nicematrix / horizontal-space } { 0 }
3646 \socket_new_plug:nnn { nicematrix / horizontal-space } { standard }
3647 { \skip_horizontal:N \g_tmpa_skip }
3648 \socket_assign_plug:nn { nicematrix / horizontal-space } { standard }
3649 \socket_new_plug:nnn { nicematrix / horizontal-space } { with-false-col }
3650 {
3651   \clist_if_in:N\TF \g_@@_false_cols_clist { \int_use:N \g_tmpa_int }
3652   {
3653     % modified 2026/09/03
3654     \skip_set:Nn \l_tmpa_skip { 0 pt+plus 1 fill }
3655     % \skip_set:Nn \l_tmpa_skip { 0 pt }
3656     \skip_add:Nn \l_tmpa_skip \l_@@_width_of_false_dim
3657     \skip_add:Nn \l_tmpa_skip \arrayrulewidth
3658     \skip_horizontal:N \l_tmpa_skip
3659   }
3660   { \skip_horizontal:N \g_tmpa_skip }
3661 }

```

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`).

```

3662 \cs_new:Npn \@@_create_col_nodes:
3663 {
3664   \crrc
3665   \int_if_zero:nT \l_@@_first_col_int
3666   {
3667     \omit
3668     \hbox_overlap_left:n
3669     {
3670       \bool_if:NT \l_@@_code_before_bool
3671       { \pgfsys@markposition { \@@_env: - col - 0 } }
3672       \pgfpicture
3673       \pgfrememberpicturepositiononpagetrue
3674       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3675       \str_if_empty:NF \l_@@_name_str
3676       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3677       \endpgfpicture
3678       \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3679     }
3680     &
3681   }
3682   \omit

```

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

```

3683   \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`).

```

3684   \int_if_zero:nTF \l_@@_first_col_int
3685   {
3686     \@@_mark_position:n { 1 }
3687     \pgfpicture
3688     \pgfrememberpicturepositiononpagetrue
3689     \pgfcoordinate { \@@_env: - col - 1 }
3690     { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
3691     \str_if_empty:NF \l_@@_name_str
3692     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3693     \endpgfpicture
3694   }
3695   {
3696     \bool_if:NT \l_@@_code_before_bool
3697     {

```

```

3698         \hbox
3699         {
3700             \skip_horizontal:n { 0.5 \arrayrulewidth }
3701             \pgfsys@markposition { \@@_env: - col - 1 }
3702             \skip_horizontal:n { -0.5 \arrayrulewidth }
3703         }
3704     }
3705     \pgfpicture
3706     \pgfrememberpicturepositiononpagetrue
3707     \pgfcoordinate { \@@_env: - col - 1 }
3708     { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3709     \@@_node_alias:n { 1 }
3710     \endpgfpicture
3711 }

```

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.

```

3712 \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
3713 \bool_if:NTF \l_@@_row_columns_width_bool
3714 { \skip_gadd:Nn \g_tmpa_skip { \box_ht_plus_dp:N \@arstrutbox } }
3715 {
3716     \bool_lazy_or:nnT
3717     { \l_@@_auto_columns_width_bool }
3718     { \dim_compare_p:nNn \l_@@_columns_width_dim > \c_zero_dim }
3719     {
3720         \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3721         \bool_lazy_and:nnTF
3722         \l_@@_auto_columns_width_bool
3723         { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3724         { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3725         { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3726     }
3727 }
3728 \int_gset:Nn \g_tmpa_int 1
3729 \socket_use:n { nicematrix / horizontal-space }
3730 \hbox
3731 {
3732     \@@_mark_position:n { 2 }
3733     \pgfpicture
3734     \pgfrememberpicturepositiononpagetrue
3735     \pgfcoordinate { \@@_env: - col - 2 }
3736     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3737     \@@_node_alias:n { 2 }
3738     \endpgfpicture
3739 }

```

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.

```

3740 \bool_if:NTF \g_@@_last_col_found_bool
3741 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3742 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3743 {
3744     &
3745     \omit
3746     \int_gincr:N \g_tmpa_int

```

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

```

3747     \socket_use:n { nicematrix / horizontal-space }
3748     \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

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

```

3749 \pgfpicture
3750 \pgfrememberpicturepositiononpagetrue
3751 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3752 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3753 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3754 \endpgfpicture
3755 }

```

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).

```

3756 \bool_lazy_or:nnF
3757 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3758 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3759 {
3760 &
3761 \omit
3762 \skip_horizontal:N \g_tmpa_skip
3763 \int_gincr:N \g_tmpa_int
3764 \bool_lazy_any:nF
3765 {
3766 \g_@@_delims_bool
3767 \l_@@_tabular_bool
3768 { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3769 \l_@@_exterior_arraycolsep_bool
3770 \l_@@_bar_at_end_of_pream_bool
3771 }
3772 { \skip_horizontal:n { - \col@sep } }
3773 \bool_if:NT \l_@@_code_before_bool
3774 {
3775 \hbox
3776 {
3777 \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.

```

3778 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3779 { \skip_horizontal:n { - \arraycolsep } }
3780 \pgfsys@markposition
3781 { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3782 \skip_horizontal:n { 0.5 \arrayrulewidth }
3783 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3784 { \skip_horizontal:N \arraycolsep }
3785 }
3786 }
3787 \pgfpicture
3788 \pgfrememberpicturepositiononpagetrue
3789 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3790 {
3791 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3792 {
3793 \pgfpoint
3794 { - 0.5 \arrayrulewidth - \arraycolsep }
3795 \c_zero_dim
3796 }
3797 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3798 }
3799 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3800 \endpgfpicture
3801 }

```

```

3802 \bool_if:NT \g_@@_last_col_found_bool

```

```

3803 {
3804   \hbox_overlap_right:n
3805   {
3806     \skip_horizontal:N \g_@@_width_last_col_dim
3807     \skip_horizontal:N \col@sep
3808     \bool_if:NT \l_@@_code_before_bool
3809     {
3810       \pgfsys@markposition
3811       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3812     }
3813     \pgfpicture
3814     \pgfrememberpicturepositiononpagetrue
3815     \pgfcoordinate
3816     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3817     \pgfpointorigin
3818     \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3819     \endpgfpicture
3820   }
3821 }
3822 }

3823 \cs_new_protected:Npn \@@_mark_position:n #1
3824 {
3825   \bool_if:NT \l_@@_code_before_bool
3826   {
3827     \hbox
3828     {
3829       \skip_horizontal:n { -0.5 \arrayrulewidth }
3830       \pgfsys@markposition { \@@_env: - col - #1 }
3831       \skip_horizontal:n { 0.5 \arrayrulewidth }
3832     }
3833   }
3834 }

3835 \cs_new_protected:Npn \@@_node_alias:n #1
3836 {
3837   \str_if_empty:NF \l_@@_name_str
3838   { \pgfnodealias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3839 }

```

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

```

3840 \tl_const:Nn \c_@@_preamble_first_col_tl
3841 {
3842   >
3843   {

```

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

```

3844   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3845   \cs_set_eq:NN \Hline \@@_Hline_in_cell:
3846   \cs_set_eq:NN \FalseRow \@@_FalseRow_in_cell:
3847   \bool_gset_true:N \g_@@_after_col_zero_bool
3848   \@@_begin_of_row:
3849   \hbox_set:Nw \l_@@_cell_box
3850   \@@_math_toggle:
3851   \@@_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”.

```

3852   \int_compare:nNnT \c@iRow > \c_zero_int
3853   {
3854     \bool_lazy_or:nnT

```

```

3855         { \int_compare:nNn \l_@@_last_row_int < \c_zero_int }
3856         { \int_compare:nNn \c_iRow < \l_@@_last_row_int }
3857         {
3858             \l_@@_code_for_first_col_tl
3859             \xglobal \colorlet { nicematrix-first-col } { . }
3860         }
3861     }
3862 }

```

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`.

```

3863     1
3864     <
3865     {
3866         \@@_math_toggle:
3867         \hbox_set_end:
3868         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3869         \@@_adjust_size_box:
3870         \@@_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.

```

3871         \dim_gset:Nn \g_@@_width_first_col_dim
3872         { \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.

```

3873         \hbox_overlap_left:n
3874         {
3875             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3876             \@@_node_cell:
3877             { \box_use_drop:N \l_@@_cell_box }
3878             \skip_horizontal:N \l_@@_left_delim_dim
3879             \skip_horizontal:N \l_@@_left_margin_dim
3880             \skip_horizontal:N \l_@@_extra_left_margin_dim
3881         }
3882         \bool_gset_false:N \g_@@_empty_cell_bool
3883         \skip_horizontal:n { -2 \col@sep }
3884     }
3885 }

```

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

```

3886 \tl_const:Nn \c_@@_preamble_last_col_tl
3887 {
3888     >
3889     {
3890         \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).

```

3891         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3892         \cs_set_eq:NN \Hline \@@_Hline_in_cell:
3893         \cs_set_eq:NN \FalseRow \@@_FalseRow_in_cell:

```

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

```

3894         \bool_gset_true:N \g_@@_last_col_found_bool
3895         \int_gincr:N \c@jCol
3896         \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3897         \hbox_set:Nw \l_@@_cell_box
3898         \@@_math_toggle:
3899         \@@_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”.

```

3900         \int_compare:nNnT \c_iRow > \c_zero_int
3901         {
3902             \bool_lazy_or:nnT

```

```

3903         { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3904         { \int_compare_p:nNn \c_iRow < \l_@@_last_row_int }
3905         {
3906             \l_@@_code_for_last_col_tl
3907             \xglobal \colorlet { nicematrix-last-col } { . }
3908         }
3909     }
3910 }
3911 1
3912 <
3913 {
3914     \@@_math_toggle:
3915     \hbox_set_end:
3916     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3917     \@@_adjust_size_box:
3918     \@@_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.

```

3919     \dim_gset:Nn \g_@@_width_last_col_dim
3920     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3921     \skip_horizontal:n { -2 \col@sep }

```

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

```

3922     \hbox_overlap_right:n
3923     {
3924         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3925         {
3926             \skip_horizontal:N \l_@@_right_delim_dim
3927             \skip_horizontal:N \l_@@_right_margin_dim
3928             \skip_horizontal:N \l_@@_extra_right_margin_dim
3929             \@@_node_cell:
3930         }
3931     }
3932     \bool_gset_false:N \g_@@_empty_cell_bool
3933 }
3934 }

```

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

```

3935 \NewDocumentEnvironment { NiceArray } { }
3936 {
3937     \bool_gset_false:N \g_@@_delims_bool
3938     \str_if_empty:NT \g_@@_name_env_str
3939     { \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).

```

3940     \NiceArrayWithDelims . .
3941 }
3942 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

3943 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3944 {
3945     \NewDocumentEnvironment { #1 NiceArray } { }
3946     {
3947         \bool_gset_true:N \g_@@_delims_bool
3948         \str_if_empty:NT \g_@@_name_env_str
3949         { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3950         \@@_test_if_math_mode:
3951         \NiceArrayWithDelims #2 #3
3952     }

```

```

3953     { \endNiceArrayWithDelims }
3954 }
3955 \@@_def_env:NNN p (      )
3956 \@@_def_env:NNN b [      ]
3957 \@@_def_env:NNN B \{      \}
3958 \@@_def_env:NNN v \vert \vert
3959 \@@_def_env:NNN V \Vert \Vert

```

13 The environment {NiceMatrix} and its variants

13.1 Definition of {pNiceMatrix}

```

3960 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3961 {
3962   \bool_set_false:N \l_@@_preamble_bool
3963   \tl_clear:N \l_tmpa_tl
3964   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3965     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3966   \tl_put_right:Nn \l_tmpa_tl
3967   {
3968     *
3969     {
3970       \int_case:nnF \l_@@_last_col_int
3971       {
3972         { -2 } { \cMaxMatrixCols }
3973         { -1 } { \int_eval:n { \cMaxMatrixCols + 1 } }

```

The value 0 can't occur here since we are in a matrix (which is an environment without preamble).

```

3974       }
3975       { \int_eval:n { \l_@@_last_col_int - 1 } }
3976     }
3977     { #2 }
3978   }
3979   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3980   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3981 }
3982 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3983 \clist_map_inline:nn { p , b , B , v , V }
3984 {
3985   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3986   {
3987     \bool_gset_true:N \g_@@_delims_bool
3988     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3989     \int_if_zero:nT \l_@@_last_col_int
3990     {
3991       \bool_set_true:N \l_@@_last_col_without_value_bool
3992       \int_set:Nn \l_@@_last_col_int { -1 }
3993     }
3994     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3995     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3996   }
3997   { \use:c { end #1 NiceArray } }
3998 }

```

We define also an environment {NiceMatrix}

```

3999 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
4000 {
4001   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }

```

```

4002 \int_if_zero:nT \l_@@_last_col_int
4003 {
4004     \bool_set_true:N \l_@@_last_col_without_value_bool
4005     \int_set:Nn \l_@@_last_col_int { -1 }
4006 }
4007 \keys_set:nn { nicematrix / NiceMatrix } { #1 }
4008 \bool_lazy_or:nnT
4009     \l_@@_except_borders_bool
4010     { \clist_if_empty_p:N \l_@@_vlines_clist }
4011     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
4012 \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
4013 }
4014 { \endNiceArray }

```

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

```

4015 \cs_new_protected:Npn \@@_NotEmpty:
4016 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

13.2 The key renew-matrix

```

4017 \cs_set_protected:Npn \@@_renew_matrix:
4018 {
4019     \tl_map_inline:nn { pvVbB }
4020     { \RenewEnvironmentCopy { ##1matrix } { ##1NiceMatrix } }
4021 }

```

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

```

4022 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
4023 {

```

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`.

```

4024     \dim_compare:nNnT\l_@@_width_dim = \c_zero_dim
4025     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
4026     \str_gset:Nn \g_@@_name_env_str { NiceTabular }
4027     \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
4028     \tl_if_empty:NF \l_@@_short_caption_tl
4029     {
4030         \tl_if_empty:NT \l_@@_caption_tl
4031         {
4032             \@@_error_or_warning:n { short-caption-without~caption }
4033             \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
4034         }
4035     }
4036     \tl_if_empty:NF \l_@@_label_tl
4037     {
4038         \tl_if_empty:NT \l_@@_caption_tl
4039         { \@@_error_or_warning:n { label~without~caption } }
4040     }
4041     \NewDocumentEnvironment { TabularNote } { b }
4042     {
4043         \bool_if:NTF \l_@@_in_code_after_bool
4044         { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
4045         {
4046             \tl_if_empty:NF \g_@@_tabularnote_tl
4047             { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
4048             \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
4049         }
4050     }
4051     { }

```

```

4052 \@@_settings_for_tabular:
4053 \NiceArray { #2 }
4054 }
4055 { \endNiceArray }
4056 \cs_new_protected:Npn \@@_settings_for_tabular:
4057 {
4058 \bool_set_true:N \l_@@_tabular_bool
4059 \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
4060 \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
4061 \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
4062 }

4063 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
4064 {
4065 \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
4066 \dim_set:Nn \l_@@_width_dim { #1 }
4067 \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
4068 \@@_settings_for_tabular:
4069 \NiceArray { #3 }
4070 }
4071 {
4072 \endNiceArray
4073 \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
4074 { \@@_error_or_warning:n { NiceTabularX~without~X } }
4075 }

4076 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
4077 {
4078 \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
4079 \dim_set:Nn \l_@@_tabular_width_dim { #1 }
4080 \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
4081 \@@_settings_for_tabular:
4082 \NiceArray { #3 }
4083 }
4084 { \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`).

```

4085 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
4086 {
4087 \bool_lazy_all:nT
4088 {
4089 { \dim_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
4090 { \l_@@_hvlines_bool }
4091 { ! \g_@@_delims_bool }
4092 { ! \l_@@_except_borders_bool }
4093 }
4094 {
4095 \bool_set_true:N \l_@@_except_borders_bool
4096 \clist_if_empty:NF \l_@@_corners_clist
4097 { \@@_error:n { hvlines,~rounded-corners~and~corners } }
4098 \tl_gput_right:Nn \g_@@_rules_tl
4099 {
4100 \@@_stroke_block:nnnnn
4101 {
4102 rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,

```

```

4103         draw = \l_@@_rules_color_tl
4104     }
4105     { 1 } { 1 } { \int_use:N \c@iRow } { \int_use:N \c@jCol }
4106 }
4107 }
4108 }

```

```

4109 \cs_new_protected:Npn \@@_after_array:
4110 {
4111     \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.

```

4112     \bool_if:NT \g_@@_last_col_found_bool
4113     { \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`.

```

4114     \bool_if:NT \l_@@_last_col_without_value_bool
4115     { \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.

```

4116     \bool_if:NT \l_@@_last_row_without_value_bool
4117     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

4118     \tl_gput_right:Ne \g_@@_aux_tl
4119     {
4120         \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
4121         {
4122             \int_use:N \l_@@_first_row_int ,
4123             \int_use:N \c@iRow ,
4124             \int_use:N \g_@@_row_total_int ,
4125             \int_use:N \l_@@_first_col_int ,
4126             \int_use:N \c@jCol ,
4127             \int_use:N \g_@@_col_total_int
4128         }
4129     }

4130     \clist_if_empty:NF \g_@@_false_cols_clist
4131     {
4132         \tl_gput_right:Ne \g_@@_aux_tl
4133         {
4134             \clist_set:Nn \exp_not:N \g_@@_false_cols_clist
4135             { \g_@@_false_cols_clist }
4136         }
4137     }

4138     \clist_if_empty:NF \g_@@_false_rows_clist
4139     {
4140         \tl_gput_right:Ne \g_@@_aux_tl
4141         {
4142             \clist_set:Nn \exp_not:N \g_@@_false_rows_clist
4143             { \g_@@_false_rows_clist }
4144         }
4145     }

4146     \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`).

```

4147 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
4148 {
4149   \tl_gput_right:Ne \g_@@_aux_tl
4150   {
4151     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
4152     { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }
4153   }
4154 }
4155 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
4156 {
4157   \tl_gput_right:Ne \g_@@_aux_tl
4158   {
4159     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
4160     { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
4161     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
4162     { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
4163   }
4164 }
```

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

```

4165 \@@_create_diag_nodes:
```

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

```

4166 \pgfpicture
4167 \@@_create_aliases_last:
4168 \str_if_empty:NF \l_@@_name_str \@@_create_alias_nodes:
4169 \endpgfpicture

4170 \bool_lazy_and:nnF
4171 { \clist_if_empty_p:N \g_@@_false_cols_clist }
4172 { \clist_if_empty_p:N \g_@@_false_rows_clist }
4173 {
4174   \@@_mark_blocks:
4175   \@@_treat_false_cols:
4176   \@@_treat_false_rows:
4177 }
```

By default, the diagonal lines will be parallelized¹³. 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.

```

4178 \bool_if:NT \l_@@_parallelize_diags_bool
4179 {
4180   \int_gzero:N \g_@@_ddots_int
4181   \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.

```

4182 \dim_gzero:N \g_@@_delta_x_one_dim
4183 \dim_gzero:N \g_@@_delta_y_one_dim
4184 \dim_gzero:N \g_@@_delta_x_two_dim
4185 \dim_gzero:N \g_@@_delta_y_two_dim
4186 }

4187 \bool_set_false:N \l_@@_initial_open_bool
4188 \bool_set_false:N \l_@@_final_open_bool
```

¹³It's possible to use the option `parallelize-diags` to disable this parallelization.

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.

```
4189 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }
```

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

```
4190 \@@_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` but (for efficiency) they will also be marked directly in the main hash table of TeX.

```
4191 \clist_if_empty:NF \l_@@_corners_clist
4192 {
4193   \bool_if:NTF \l_@@_no_cell_nodes_bool
4194   { \@@_error:n { corners~with~no~cell~nodes } }
4195   \@@_compute_corners:
4196 }
```

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`.

```
4197 \seq_gconcat:NNN \g_@@_pos_of_blocks_seq
4198   \g_@@_pos_of_blocks_seq
4199   \g_@@_future_pos_of_blocks_seq
4200 \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-*}`).

```
4201 \@@_adjust_pos_of_blocks_seq:
4202 \@@_deal_with_rounded_corners:
4203 \legacy_if:nF { measuring@ } \@@_draw_rules:
4204 \tl_gclear:N \g_@@_rules_tl
```

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

```
4205 \IfPackageLoadedT { tikz }
4206 {
4207   \tikzset
4208   {
4209     every-picture / .style =
4210     {
4211       overlay ,
4212       remember-picture ,
4213       name-prefix = \@@_env: -
4214     }
4215   }
4216 }
4217 \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
4218 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
4219 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
4220 \cs_set_eq:NN \OverBrace \@@_OverBrace
4221 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
4222 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
4223 \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.

```
4224 \legacy_if:nF { measuring@ } \g_@@_pre_code_after_tl
4225 \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.

```
4226 \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).

```
4227 \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`).

```
4228 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
4229 { \@@_rescan_for_spanish:N \g_nicematrix_code_after_tl }
4230 \bool_set_true:N \l_@@_in_code_after_bool
4231 \@@_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:`.

```
4232 \bool_set_true:N \l_@@_in_code_after_bool
4233 \if_mode_math:
4234 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4235 \scan_stop:
4236 \else:
4237 $ % $
4238 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4239 \scan_stop:
4240 $ % $
4241 \fi:
4242 \tl_gclear:N \g_nicematrix_code_after_tl
4243 \clist_if_empty:NF \g_@@_col_with_trees_clist \@@_draw_trees:
4244 \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.

```
4245 \seq_if_empty:NF \g_@@_rowlistcolors_seq \@@_clear_rowlistcolors_seq:
4246 \tl_if_empty:NF \g_@@_pre_code_before_tl
4247 {
4248 \tl_gput_right:Ne \g_@@_aux_tl
4249 {
4250 \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
4251 { \exp_not:o \g_@@_pre_code_before_tl }
4252 }
4253 \tl_gclear:N \g_@@_pre_code_before_tl
4254 }
4255 \tl_if_empty:NF \g_nicematrix_code_before_tl
4256 {
4257 \tl_gput_right:Ne \g_@@_aux_tl
4258 {
4259 \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
4260 { \exp_not:o \g_nicematrix_code_before_tl }
4261 }
4262 \tl_gclear:N \g_nicematrix_code_before_tl
4263 }

4264 \str_gclear:N \g_@@_name_env_str
4265 \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the array¹⁴. 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 use the following instruction which is in the patch of the end of arrays done by `colortbl`.

```

4266     \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
4267 }

4268 \cs_new_protected:Npn \@@_test_false_columns:
4269 {
4270     \clist_map_inline:Nn \g_@@_false_cols_clist
4271     {
4272         \int_step_inline:nnn { \l_@@_first_row_int } { \arabic { iRow } }
4273         {
4274             \cs_if_exist:cT
4275             { pgf @ sh @ ns @ \@@_env: - ####1 - ##1 }
4276             {
4277                 \@@_error:nnn { Not~empty~cell~in~false~column }
4278                 { ####1 }
4279                 { ##1 }
4280             }
4281         }
4282     }
4283 }

4284 \cs_new_protected:Npn \@@_treat_false_rows:
4285 { \clist_map_function:NN \g_@@_false_rows_clist \@@_treat_one_false_row:n }

4286 \cs_new_protected:Npn \@@_treat_one_false_row:n #1
4287 {
4288     \int_zero:N \l_tmpa_int
4289     \int_step_inline:nn \c@jCol
4290     {
4291         \int_if_zero:nTF \l_tmpa_int
4292         {
4293             \@@_if_in_block:nnF { #1 } { ##1 }
4294             { \int_set:Nn \l_tmpa_int { ##1 } }
4295         }
4296         {
4297             \@@_if_in_block:nnT { #1 } { ##1 }
4298             {
4299                 \@@_false_h_segment:nee
4300                 { #1 }
4301                 { \int_use:N \l_tmpa_int }
4302                 { \int_eval:n { ##1 - 1 } }
4303                 \int_zero:N \l_tmpa_int
4304             }
4305         }
4306     }
4307     \int_compare:nNnT \l_tmpa_int > 0
4308     {
4309         \clist_if_in:NneTF \g_@@_false_cols_clist { \int_use:N \c@jCol }
4310         { \int_set_eq:NN \l_tmpb_int \c@jCol }
4311         { \int_set:Nn \l_tmpb_int { \c@jCol + 1 } }
4312         \@@_false_h_segment:nee { #1 }
4313         { \int_use:N \l_tmpa_int }
4314         { \int_use:N \l_tmpb_int }
4315     }
4316 }

```

¹⁴e.g. `\color[rgb]{0.5,0.5,0}`

```

4317 \cs_new_protected:Npn \@@_false_h_segment:nnn #1 #2 #3
4318 {

```

The command `\clist_if_in:NnTF` is *not* expandable and that's why we have to transit by a variable `\l_tmpa_int`.

```

4319   \int_compare:nNnTF { #2 } = 1
4320   {
4321     \clist_if_in:NnTF \g_@@_false_cols_clist { 1 }
4322     { \int_set:Nn \l_tmpa_int 1 }
4323     { \int_zero:N \l_tmpa_int }
4324   }
4325   { \int_set:Nn \l_tmpa_int { #2 } }
4326   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
4327   {
4328     { #1 }
4329     { \int_use:N \l_tmpa_int }
4330     { #1 }
4331     { #3 }
4332     { }
4333   }
4334   \tl_gput_right:Ne \g_@@_pre_code_before_tl
4335   { \@@_rectanglecolor { \l_@@_color_of_false_tl } { #1 - #2 } { #1 - #3 } }
4336 }
4337 \cs_generate_variant:Nn \@@_false_h_segment:nnn { n e e }
4338 \cs_new_protected:Npn \@@_treat_false_cols:
4339 { \clist_map_function:NN \g_@@_false_cols_clist \@@_treat_one_false_col:n }
4340 \cs_new_protected:Npn \@@_treat_one_false_col:n #1
4341 {
4342   \int_zero:N \l_tmpa_int
4343   \int_step_inline:nn \c@iRow
4344   {
4345     \int_if_zero:nTF \l_tmpa_int
4346     {
4347       \@@_if_in_block:nnF { ##1 } { #1 }
4348       { \int_set:Nn \l_tmpa_int { ##1 } }
4349     }
4350   }
4351   {
4352     \@@_if_in_block:nnT { ##1 } { #1 }
4353     {
4354       \@@_false_v_segment:nee
4355       { #1 }
4356       { \int_use:N \l_tmpa_int }
4357       { \int_eval:n { ##1 - 1 } }
4358       \int_zero:N \l_tmpa_int
4359     }
4360   }
4361 }
4362 \int_compare:nNnT \l_tmpa_int > 0
4363 {
4364   \clist_if_in:NnTF \g_@@_false_rows_clist { \int_use:N \c@iRow }
4365   { \int_set_eq:NN \l_tmpb_int \c@iRow }
4366   { \int_set:Nn \l_tmpb_int { \c@iRow + 1 } }
4367   \@@_false_v_segment:nee { #1 }
4368   { \int_use:N \l_tmpa_int }
4369   { \int_use:N \l_tmpb_int }
4370 }
4371 }

```

```

4372 \cs_new_protected:Npn \@@_false_v_segment:nnn #1 #2 #3
4373 {
4374   \int_compare:nNnTF { #2 } = 1

```

```

4375 {
4376   \clist_if_in:NnTF \g_@@_false_rows_clist { 1 }
4377   { \int_set:Nn \l_tmpa_int 1 }
4378   { \int_zero:N \l_tmpa_int }
4379 }
4380 { \int_set:Nn \l_tmpa_int { #2 } }
4381 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
4382 {
4383   { \int_use:N \l_tmpa_int }
4384   { #1 }
4385   { #3 }
4386   { #1 }
4387   { }
4388 }
4389 \tl_gput_right:Ne \g_@@_pre_code_before_tl
4390 { \@@_rectanglecolor { \l_@@_color_of_false_tl } { #2 - #1 } { #3 - #1 } }
4391 }
4392 \cs_generate_variant:Nn \@@_false_v_segment:nnn { n e e }

```

```

4393 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
4394 {
4395   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
4396   \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.

```

4397   \dim_set:Nn \l_@@_xdots_shorten_start_dim
4398   { 0.6 \l_@@_xdots_shorten_start_dim }
4399   \dim_set:Nn \l_@@_xdots_shorten_end_dim
4400   { 0.6 \l_@@_xdots_shorten_end_dim }
4401 }

```

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`.

```

4402 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4403 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

```

```

4404 \cs_new_protected:Npn \@@_create_alias_nodes:
4405 {
4406   \int_step_inline:nn \c@iRow
4407   {
4408     \pgfnodealias
4409     { \l_@@_name_str - ##1 - last }
4410     { \@@_env: - ##1 - \int_use:N \c@jCol }
4411   }
4412   \int_step_inline:nn \c@jCol
4413   {
4414     \pgfnodealias
4415     { \l_@@_name_str - last - ##1 }
4416     { \@@_env: - \int_use:N \c@iRow - ##1 }
4417   }
4418   \pgfnodealias
4419   { \l_@@_name_str - last - last }
4420   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
4421 }

```

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).

```

4422 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4423 {
4424   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4425   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4426 }

```

The following command must *not* be protected.

```

4427 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4428 {
4429   { #1 }
4430   { #2 }
4431   {
4432     \int_compare:nNnTF { #3 } > { 98 }
4433     { \int_use:N \c@iRow }
4434     { #3 }
4435   }
4436   {
4437     \int_compare:nNnTF { #4 } > { 98 }
4438     { \int_use:N \c@jCol }
4439     { #4 }
4440   }
4441   { #5 }
4442 }

```

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).

```

4443 \AtBeginDocument
4444 {
4445   \cs_new_protected:Npe \@@_draw_dotted_lines:
4446   {
4447     \c_@@_pgfortikzpicture_tl
4448     \@@_draw_dotted_lines_i:
4449     \c_@@_endpgfortikzpicture_tl
4450   }
4451 }

```

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

```

4452 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4453 {
4454   \pgfrememberpicturerepositiononpagetrue
4455   \pgf@relevantforpicturesizefalse
4456   \g_@@_HVdotsfor_lines_tl
4457   \g_@@_Vdots_lines_tl
4458   \g_@@_Ddots_lines_tl
4459   \g_@@_Iddots_lines_tl
4460   \g_@@_Cdots_lines_tl
4461   \g_@@_Ldots_lines_tl
4462 }

4463 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4464 {
4465   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4466   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4467 }

```

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

```

4468 \pgfdeclareshape { @@_diag_node }
4469 {
4470   \savedanchor { \five }
4471   {
4472     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4473     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4474   }
4475   \anchor { 5 } { \five }
4476   \anchor { center } { \pgfpointorigin }
4477   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4478   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4479   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4480   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4481   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4482   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4483   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4484   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4485   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4486   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4487 }

```

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).

```

4488 \cs_new_protected:Npn \@_create_diag_nodes:
4489 {
4490   \pgfpicture
4491   \pgfrememberpicturepositiononpagetrue
4492   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4493   {
4494     \@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4495     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4496     \@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4497     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4498     \@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4499     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4500     \@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4501     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4502     \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.

```

4503     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4504     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4505     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4506     \str_if_empty:NF \l_@@_name_str
4507     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4508   }

```

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

```

4509     \int_set:Nn \l_tmpa_int { 1 + \int_max:nn \c@iRow \c@jCol } % modified
4510     \@_qpoint:n { row - \int_min:nn \l_tmpa_int { \c@iRow + 1 } }
4511     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4512     \@_qpoint:n { col - \int_min:nn \l_tmpa_int { \c@jCol + 1 } }
4513     \pgfcoordinate
4514     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4515     \pgfnodealias
4516     { \@@_env: - last }
4517     { \@@_env: - \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
4518     \str_if_empty:NF \l_@@_name_str
4519     {

```

```

4520     \pgfnodealias
4521     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4522     { \@@_env: - \int_use:N \l_tmpa_int }
4523     \pgfnodealias
4524     { \l_@@_name_str - last }
4525     { \@@_env: - last }
4526   }
4527 \endpgfpicture
4528 }

```

16 We draw the dotted lines

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

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

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

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

This command computes:

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

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

```

\cs_new_protected:Npn \@@_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.

```

4529 \cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
4530 {

```

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

```

4531 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4532 \l_@@_initial_i_int = #1
4533 \l_@@_initial_j_int = #2
4534 \l_@@_final_i_int = #1
4535 \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.

```

4536 \let \l_@@_stop_loop_bool \c_false_bool
4537 \bool_do_until:Nn \l_@@_stop_loop_bool
4538 {

```

We test if we are still in the matrix.

```

4539 \advance \l_@@_final_i_int by #3
4540 \advance \l_@@_final_j_int by #4
4541 \let \l_@@_final_open_bool \c_false_bool
4542 \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4543 \if_int_compare:w #3 = \c_one_int
4544 \let \l_@@_final_open_bool \c_true_bool
4545 \else:
4546 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4547 \let \l_@@_final_open_bool \c_true_bool
4548 \fi:
4549 \fi:
4550 \else:
4551 \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4552 \if_int_compare:w #4 = -1
4553 \let \l_@@_final_open_bool \c_true_bool
4554 \fi:
4555 \else:
4556 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4557 \if_int_compare:w #4 = \c_one_int
4558 \let \l_@@_final_open_bool \c_true_bool
4559 \fi:
4560 \fi:
4561 \fi:
4562 \fi:
4563 \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.

```

4564 {

```

We do a step backwards.

```

4565 \advance \l_@@_final_i_int by - #3
4566 \advance \l_@@_final_j_int by - #4
4567 \let \l_@@_stop_loop_bool \c_true_bool
4568 }

```

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`.

```

4569     {
4570         \cs_if_exist:cTF
4571         {
4572             @@ _ dotted _
4573             \int_use:N \l_@@_final_i_int -
4574             \int_use:N \l_@@_final_j_int
4575         }
4576         {
4577             \advance \l_@@_final_i_int by - #3
4578             \advance \l_@@_final_j_int by - #4
4579             \let \l_@@_final_open_bool \c_true_bool
4580             \let \l_@@_stop_loop_bool \c_true_bool
4581         }
4582     }
4583     \cs_if_exist:cTF
4584     {
4585         pgf @ sh @ ns @ \@@_env:
4586         - \int_use:N \l_@@_final_i_int
4587         - \int_use:N \l_@@_final_j_int
4588     }
4589     { \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.

```

4590     {
4591         \cs_set_nopar:cpn
4592         {
4593             @@ _ dotted _
4594             \int_use:N \l_@@_final_i_int -
4595             \int_use:N \l_@@_final_j_int
4596         }
4597         { }
4598     }
4599 }
4600 }
4601 }

```

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

```

4602     \let \l_@@_stop_loop_bool \c_false_bool

```

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

```

4603     \l_tmpa_int = \l_@@_col_min_int
4604     \advance \l_tmpa_int by -1
4605     \bool_do_until:Nn \l_@@_stop_loop_bool
4606     {
4607         \advance \l_@@_initial_i_int by - #3
4608         \advance \l_@@_initial_j_int by - #4
4609         \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.

```

4610     \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4611     \if_int_compare:w #3 = \c_one_int
4612         \let \l_@@_initial_open_bool \c_true_bool
4613     \else:

```

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

```

4614         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4615         \let \l_@@_initial_open_bool \c_true_bool
4616     \fi:
4617 \fi:
4618 \else:
4619     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4620     \if_int_compare:w #4 = \c_one_int
4621     \let \l_@@_initial_open_bool \c_true_bool
4622     \fi:
4623 \else:
4624     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4625     \if_int_compare:w #4 = -1
4626     \let \l_@@_initial_open_bool \c_true_bool
4627     \fi:
4628 \fi:
4629 \fi:
4630 \fi:
4631 \bool_if:NTF \l_@@_initial_open_bool
4632 {
4633     \advance \l_@@_initial_i_int by #3
4634     \advance \l_@@_initial_j_int by #4
4635     \let \l_@@_stop_loop_bool \c_true_bool
4636 }
4637 {
4638     \cs_if_exist:cTF
4639     {
4640         @@ _ dotted _
4641         \int_use:N \l_@@_initial_i_int -
4642         \int_use:N \l_@@_initial_j_int
4643     }
4644     {
4645         \advance \l_@@_initial_i_int by #3
4646         \advance \l_@@_initial_j_int by #4
4647         \let \l_@@_initial_open_bool \c_true_bool
4648         \let \l_@@_stop_loop_bool \c_true_bool
4649     }
4650     {
4651         \cs_if_exist:cTF
4652         {
4653             pgf @ sh @ ns @ \@@_env:
4654             - \int_use:N \l_@@_initial_i_int
4655             - \int_use:N \l_@@_initial_j_int
4656         }
4657         { \let \l_@@_stop_loop_bool \c_true_bool }
4658         {
4659             \cs_set_nopar:cpn
4660             {
4661                 @@ _ dotted _
4662                 \int_use:N \l_@@_initial_i_int -
4663                 \int_use:N \l_@@_initial_j_int
4664             }
4665             { }
4666         }
4667     }
4668 }
4669 }
```

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.

```

4670 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4671 {
4672     { \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`.

```

4673     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4674     { \int_use:N \l_@@_final_i_int }
4675     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4676     { }
4677   }
4678 }

```

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 know 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.

```

4679 \cs_new_protected:Npn \@@_open_shorten:
4680 {
4681   \bool_if:NT \l_@@_initial_open_bool
4682     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4683   \bool_if:NT \l_@@_final_open_bool
4684     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4685 }

```

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).

```

4686 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4687 {
4688   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4689   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4690   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4691   \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.

```

4692   \seq_if_empty:NTF \g_@@_submatrix_seq
4693     { \cs_set_eq:NN \@@_adjust_to_submatrix:nn \use_none:nn }
4694   {

```

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`.

```

4695     \seq_map_inline:Nn \g_@@_submatrix_seq
4696       { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4697   }
4698 }

```

#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.

```

4699 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4700 {
4701     \if_int_compare:w #3 > #1
4702     \else:
4703         \if_int_compare:w #1 > #5
4704         \else:
4705             \if_int_compare:w #4 > #2
4706             \else:
4707                 \if_int_compare:w #2 > #6
4708                 \else:
4709                     \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4710                     \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4711                     \if_int_compare:w \l_@@_row_max_int > #5 \l_@@_row_max_int = #5 \fi:
4712                     \if_int_compare:w \l_@@_col_max_int > #6 \l_@@_col_max_int = #6 \fi:
4713                 \fi:
4714             \fi:
4715         \fi:
4716     \fi:
4717 }

4718 \cs_new_protected:Npn \@@_set_initial_coords:
4719 {
4720     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4721     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4722 }
4723 \cs_new_protected:Npn \@@_set_final_coords:
4724 {
4725     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4726     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4727 }
4728 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4729 {
4730     \pgfpointanchor
4731     {
4732         \@@_env:
4733         - \int_use:N \l_@@_initial_i_int
4734         - \int_use:N \l_@@_initial_j_int
4735     }
4736     { #1 }
4737     \@@_set_initial_coords:
4738 }
4739 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4740 {
4741     \pgfpointanchor
4742     {
4743         \@@_env:
4744         - \int_use:N \l_@@_final_i_int
4745         - \int_use:N \l_@@_final_j_int
4746     }
4747     { #1 }
4748     \@@_set_final_coords:
4749 }
4750 \cs_new_protected:Npn \@@_open_x_initial_dim:
4751 {
4752     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4753     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4754     {
4755         \cs_if_exist:cT

```

```

4756     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4757     {
4758       \pgfpointanchor
4759       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4760       { west }
4761       \dim_set:Nn \l_@@_x_initial_dim
4762       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4763     }
4764   }

```

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

```

4765   \dim_compare:nNtT \l_@@_x_initial_dim = \c_max_dim
4766   {
4767     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4768     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4769     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4770   }
4771 }

4772 \cs_new_protected:Npn \@@_open_x_final_dim:
4773 {
4774   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4775   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4776   {
4777     \cs_if_exist:cT
4778     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4779     {
4780       \pgfpointanchor
4781       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4782       { east }
4783       \dim_compare:nNtT \pgf@x > \l_@@_x_final_dim
4784       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4785     }
4786   }

```

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

```

4787   \dim_compare:nNtT \l_@@_x_final_dim = { - \c_max_dim }
4788   {
4789     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4790     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4791     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4792   }
4793 }

```

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.

```

4794 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4795 {
4796   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4797   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4798   {
4799     \@@_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.

```

4800   \bool_if:NT \g_@@_aux_found_bool
4801   {
4802     {
4803       \@@_open_shorten:
4804       \int_if_zero:nTF { #1 }
4805       { \color { nicematrix-first-row } }
4806     }

```

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.

```

4807         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4808         { \color { nicematrix-last-row } }
4809     }
4810     \keys_set:nn { nicematrix / xdots } { #3 }
4811     \@@_color:o \l_@@_xdots_color_tl
4812     \@@_actually_draw_Ldots:
4813 }
4814 }
4815 }
4816 }

```

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

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

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

```

4817 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4818 {
4819     \bool_if:NTF \l_@@_initial_open_bool
4820     {
4821         \@@_open_x_initial_dim:
4822         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4823         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4824     }
4825     { \@@_set_initial_coords_from_anchor:n { base~east } }
4826     \bool_if:NTF \l_@@_final_open_bool
4827     {
4828         \@@_open_x_final_dim:
4829         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4830         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4831     }
4832     { \@@_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.

```

4833     \bool_lazy_all:nTF
4834     {
4835         \l_@@_initial_open_bool
4836         \l_@@_final_open_bool
4837         { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4838     }
4839     {
4840         \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4841         \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4842     }

```

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

```

4843     {
4844         \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4845         \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4846     }
4847     \@@_draw_line:
4848 }

```

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.

```

4849 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4850 {
4851   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4852   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4853   {
4854     \@@_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.

```

4855     \bool_if:NT \g_@@_aux_found_bool
4856     {
4857       {
4858         \@@_open_shorten:
4859         \int_if_zero:nTF { #1 }
4860         { \color { nicematrix-first-row } }
4861         {

```

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.

```

4862         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4863         { \color { nicematrix-last-row } }
4864       }
4865       \keys_set:nn { nicematrix / xdots } { #3 }
4866       \@@_color:o \l_@@_xdots_color_tl
4867       \@@_actually_draw_Cdots:
4868     }
4869   }
4870 }
4871 }

```

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`.

```

4872 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4873 {
4874   \bool_if:NTF \l_@@_initial_open_bool
4875   \@@_open_x_initial_dim:
4876   { \@@_set_initial_coords_from_anchor:n { mid-east } }
4877   \bool_if:NTF \l_@@_final_open_bool
4878   \@@_open_x_final_dim:
4879   { \@@_set_final_coords_from_anchor:n { mid-west } }
4880   \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4881   {
4882     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4883     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4884     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4885     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4886     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4887   }
4888   {
4889     \bool_if:NT \l_@@_initial_open_bool
4890     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4891     \bool_if:NT \l_@@_final_open_bool

```

```

4892         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4893     }
4894     \@@_draw_line:
4895 }
4896 \cs_new_protected:Npn \@@_open_y_initial_dim:
4897 {
4898     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4899     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4900     {
4901         \cs_if_exist:cT
4902         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4903         {
4904             \pgfpointanchor
4905             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4906             { north }
4907             \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4908             { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4909         }
4910     }
4911     \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4912     {
4913         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4914         \dim_set:Nn \l_@@_y_initial_dim
4915         {
4916             \fp_to_dim:n
4917             {
4918                 \pgf@y
4919                 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4920             }
4921         }
4922     }
4923 }
4924 \cs_new_protected:Npn \@@_open_y_final_dim:
4925 {
4926     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4927     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4928     {
4929         \cs_if_exist:cT
4930         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4931         {
4932             \pgfpointanchor
4933             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4934             { south }
4935             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4936             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4937         }
4938     }
4939     \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4940     {
4941         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4942         \dim_set:Nn \l_@@_y_final_dim
4943         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4944     }
4945 }

```

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.

```

4946 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4947 {
4948     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4949     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4950     {
4951         \@@_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.

```

4952     \bool_if:NT \g_@@_aux_found_bool
4953     {
4954     {
4955         \@@_open_shorten:
4956         \int_if_zero:nTF { #2 }
4957         { \color { nicematrix-first-col } }
4958         {
4959             \int_compare:nNnT { #2 } = \l_@@_last_col_int
4960             { \color { nicematrix-last-col } }
4961         }
4962         \keys_set:nn { nicematrix / xdots } { #3 }
4963         \@@_color:o \l_@@_xdots_color_tl
4964         \bool_if:NTF \l_@@_Vbrace_bool
4965         \@@_actually_draw_Vbrace:
4966         \@@_actually_draw_Vdots:
4967     }
4968     }
4969 }
4970 }
```

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`.

```

4971 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4972 {
4973     \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4974     \@@_actually_draw_Vdots_i:
4975     \@@_actually_draw_Vdots_ii:
4976     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4977     \@@_draw_line:
4978 }
```

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

```

4979 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4980 {
4981     \@@_open_y_initial_dim:
4982     \@@_open_y_final_dim:
4983     \int_if_zero:nTF \l_@@_initial_j_int
```

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

```

4984     {
4985         \@@_qpoint:n { col - 1 }
4986         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4987         \dim_sub:Nn \l_@@_x_initial_dim
4988         { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4989     }
4990     {
4991         \bool_lazy_and:nnTF
4992         { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4993         { \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”.

```

4994     {
4995         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4996         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4997         \dim_add:Nn \l_@@_x_initial_dim
4998             { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4999     }

```

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

```

5000     {
5001         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
5002         \dim_set_eq:NN \l_tmpa_dim \pgf@x
5003         \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
5004         \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
5005     }
5006 }
5007 }

```

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.

```

5008 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
5009 {
5010     \bool_set_false:N \l_tmpa_bool
5011     \bool_if:NF \l_@@_initial_open_bool
5012     {
5013         \bool_if:NF \l_@@_final_open_bool
5014         {
5015             \@@_set_initial_coords_from_anchor:n { south-west }
5016             \@@_set_final_coords_from_anchor:n { north-west }
5017             \bool_set:Nn \l_tmpa_bool
5018                 { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
5019         }
5020     }

```

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

```

5021     \bool_if:NTF \l_@@_initial_open_bool
5022     {
5023         \@@_open_y_initial_dim:
5024         \@@_set_final_coords_from_anchor:n { north }
5025         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
5026     }
5027     {
5028         \@@_set_initial_coords_from_anchor:n { south }
5029         \bool_if:NTF \l_@@_final_open_bool
5030         \@@_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.

```

5031     {
5032         \@@_set_final_coords_from_anchor:n { north }
5033         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
5034         {
5035             \dim_set:Nn \l_@@_x_initial_dim
5036             {
5037                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
5038                     \l_@@_x_initial_dim \l_@@_x_final_dim
5039             }
5040         }
5041     }
5042 }
5043 }

```

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`.

```

5044 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
5045   {
5046     \bool_if:NTF \l_@@_initial_open_bool
5047       \@@_open_y_initial_dim:
5048       { \@@_set_initial_coords_from_anchor:n { south } }
5049     \bool_if:NTF \l_@@_final_open_bool
5050       \@@_open_y_final_dim:
5051       { \@@_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.

```

5052   \int_if_zero:nTF \l_@@_initial_j_int
5053   {
5054     \@@_qpoint:n { col - 1 }
5055     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5056     \dim_sub:Nn \l_@@_x_initial_dim
5057     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
5058   }

```

Elsewhere, the brace must be drawn left flush.

```

5059   {
5060     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
5061     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5062     \dim_add:Nn \l_@@_x_initial_dim
5063     { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
5064   }

```

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

```

5065   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
5066   \@@_draw_line:
5067 }

```

```

5068 \cs_new:Npn \@@_colsep:
5069   { \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.

```

5070 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
5071   {
5072     \@@_adjust_to_submatrix:nn { #1 } { #2 }
5073     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
5074     {
5075       \@@_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.

```

5076     \bool_if:NT \g_@@_aux_found_bool
5077     {
5078     {
5079         \@@_open_shorten:
5080         \keys_set:nn { nicematrix / xdots } { #3 }
5081         \@@_color:o \l_@@_xdots_color_tl
5082         \@@_actually_draw_Ddots:
5083     }
5084     }
5085 }
5086 }
```

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`.

```

5087 \cs_new_protected:Npn \@@_actually_draw_Ddots:
5088 {
5089     \bool_if:NTF \l_@@_initial_open_bool
5090     {
5091         \@@_open_y_initial_dim:
5092         \@@_open_x_initial_dim:
5093     }
5094     { \@@_set_initial_coords_from_anchor:n { south~east } }
5095     \bool_if:NTF \l_@@_final_open_bool
5096     {
5097         \@@_open_x_final_dim:
5098         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5099     }
5100     { \@@_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.

```

5101     \bool_if:NT \l_@@_parallelize_diags_bool
5102     {
5103         \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).

```

5104         \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.

```

5105     {
5106         \dim_gset:Nn \g_@@_delta_x_one_dim
5107         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
5108         \dim_gset:Nn \g_@@_delta_y_one_dim
5109         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
5110     }
```

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`.

```

5111     {
5112         \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
5113         {
5114             \dim_set:Nn \l_@@_y_final_dim
5115             {
5116                 \l_@@_y_initial_dim +
5117                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5118                 \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
5119             }
5120         }
5121     }
5122 }
5123 \@@_draw_line:
5124 }

```

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.

```

5125 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
5126 {
5127     \@@_adjust_to_submatrix:nn { #1 } { #2 }
5128     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
5129     {
5130         \@@_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.

```

5131         \bool_if:NT \g_@@_aux_found_bool
5132         {
5133             {
5134                 \@@_open_shorten:
5135                 \keys_set:nn { nicematrix / xdots } { #3 }
5136                 \@@_color:o \l_@@_xdots_color_tl
5137                 \@@_actually_draw_Iddots:
5138             }
5139         }
5140     }
5141 }

```

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`.

```

5142 \cs_new_protected:Npn \@@_actually_draw_Iddots:
5143 {
5144     \bool_if:NTF \l_@@_initial_open_bool
5145     {
5146         \@@_open_y_initial_dim:
5147         \@@_open_x_initial_dim:
5148     }
5149     { \@@_set_initial_coords_from_anchor:n { south-west } }
5150     \bool_if:NTF \l_@@_final_open_bool

```

```

5151 {
5152   \l_@@_open_y_final_dim:
5153   \l_@@_open_x_final_dim:
5154 }
5155 { \l_@@_set_final_coords_from_anchor:n { north-east } }
5156 \bool_if:NT \l_@@_parallelize_diags_bool
5157 {
5158   \int_gincr:N \g_@@_iddots_int
5159   \int_compare:nNnTF \g_@@_iddots_int = 1
5160   {
5161     \dim_gset:Nn \g_@@_delta_x_two_dim
5162     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
5163     \dim_gset:Nn \g_@@_delta_y_two_dim
5164     { \l_@@_y_final_dim - \l_@@_y_initial_dim }
5165   }
5166   {
5167     \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
5168     {
5169       \dim_set:Nn \l_@@_y_final_dim
5170       {
5171         \l_@@_y_initial_dim +
5172         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5173         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
5174       }
5175     }
5176   }
5177 }
5178 \l_@@_draw_line:
5179 }

```

17 The actual instructions for drawing the dotted lines with TikZ

The command `\l_@@_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`

```

5180 \cs_new_protected:Npn \l_@@_draw_line:
5181 {
5182   \pgfrememberpicturepositiononpagetrue
5183   \pgf@relevantforpicturesizefalse
5184   \bool_lazy_or:nnTF
5185     \l_@@_dotted_bool
5186     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
5187     \l_@@_draw_standard_dotted_line:
5188     \l_@@_draw_unstandard_dotted_line:
5189 }

```

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.

```

5190 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
5191 {
5192   \begin { scope }
5193     \@@_draw_unstandard_dotted_line:o
5194     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
5195 }

```

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.

```

5196 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
5197 {
5198   \@@_draw_unstandard_dotted_line:nooo
5199   { #1 }
5200   \l_@@_xdots_up_tl
5201   \l_@@_xdots_down_tl
5202   \l_@@_xdots_middle_tl
5203 }
5204 \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.

```

5205 \AtBeginDocument
5206 {
5207   \IfPackageLoadedT { tikz }
5208   {
5209     \tikzset
5210     {
5211       @@_node_above / .style = { sloped , above } ,
5212       @@_node_below / .style = { sloped , below } ,
5213       @@_node_middle / .style =
5214       {
5215         sloped ,
5216         inner~sep = \c_@@_innersep_middle_dim
5217       }
5218     }
5219   }
5220 }

5221 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
5222 {

```

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.

```

5223 \dim_zero_new:N \l_@@_l_dim
5224 \dim_set:Nn \l_@@_l_dim
5225 {
5226   \fp_to_dim:n
5227   {
5228     sqrt
5229     (
5230       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5231       +
5232       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5233     )
5234   }
5235 }

```

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.

```

5236   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
5237   {
5238     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
5239     \@@_draw_unstandard_dotted_line_i:
5240   }

```

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

```

5241   \bool_if:NT \l_@@_xdots_h_labels_bool
5242   {
5243     \tikzset
5244     {
5245       @@_node_above / .style = { auto = left } ,
5246       @@_node_below / .style = { auto = right } ,
5247       @@_node_middle / .style = { innersep = \c_@@_innersep_middle_dim }
5248     }
5249   }
5250   \tl_if_empty:nF { #4 }
5251   { \tikzset { @@_node_middle / .append~style = { fill = white } } }
5252   \dim_zero:N \l_tmpa_dim
5253   \dim_zero:N \l_tmpb_dim
5254   \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
5255   {

```

We test whether the brace is vertical or horizontal.

```

5256     \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
5257     { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
5258     { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
5259   }
5260   {
5261     \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
5262     {
5263       \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
5264       { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
5265       { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
5266     }
5267   }
5268   \use:e
5269   {
5270     \exp_not:N \begin { scope }
5271     [ shift = {(\dim_use:N \l_tmpa_dim, \dim_use:N \l_tmpb_dim)} ]
5272   }
5273   \draw
5274   [ #1 ]
5275   ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
5276   -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
5277   node [ @@_node_below ] { $ \scriptstyle #3 $ }
5278   node [ @@_node_above ] { $ \scriptstyle #2 $ }
5279   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
5280   \end { scope }
5281   \end { scope }
5282 }
5283 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
5284 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
5285 {
5286   \dim_set:Nn \l_tmpa_dim
5287   {
5288     \l_@@_x_initial_dim
5289     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5290     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim

```

```

5291     }
5292     \dim_set:Nn \l_tmpb_dim
5293     {
5294         \l_@@_y_initial_dim
5295         + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5296         * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
5297     }
5298     \dim_set:Nn \l_@@_tmpc_dim
5299     {
5300         \l_@@_x_final_dim
5301         - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5302         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5303     }
5304     \dim_set:Nn \l_@@_tmpd_dim
5305     {
5306         \l_@@_y_final_dim
5307         - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5308         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5309     }
5310     \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
5311     \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
5312     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
5313     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
5314 }

```

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

```

5315 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
5316 {
5317 {

```

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.

```

5318     \dim_zero_new:N \l_@@_l_dim
5319     \dim_set:Nn \l_@@_l_dim
5320     {
5321         \fp_to_dim:n
5322         {
5323             sqrt
5324             (
5325                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5326                 +
5327                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5328             )
5329         }
5330     }

```

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.

```

5331     \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
5332     {
5333         \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
5334         \@@_draw_standard_dotted_line_i:
5335     }
5336 }
5337 \bool_lazy_all:nF
5338 {
5339     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
5340     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
5341     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
5342 }

```

```

5343 \@@_labels_standard_dotted_line:
5344 }
5345 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
5346 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
5347 {

```

The number of dots will be $\l_1\text{tmpa_int} + 1$.

```

5348 \int_set:Nn \l_1tmpa_int
5349 {
5350 \dim_ratio:nn
5351 {
5352 \l_@@_l_dim
5353 - \l_@@_xdots_shorten_start_dim
5354 - \l_@@_xdots_shorten_end_dim
5355 }
5356 \l_@@_xdots_inter_dim
5357 }

```

The dimensions $\l_1\text{tmpa_dim}$ and $\l_1\text{tmpb_dim}$ are the coordinates of the vector between two dots in the dotted line.

```

5358 \dim_set:Nn \l_1tmpa_dim
5359 {
5360 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5361 \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5362 }
5363 \dim_set:Nn \l_1tmpb_dim
5364 {
5365 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5366 \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5367 }

```

In the loop over the dots, the dimensions $\l_1\text{@@_x_initial_dim}$ and $\l_1\text{@@_y_initial_dim}$ will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

5368 \dim_gadd:Nn \l_@@_x_initial_dim
5369 {
5370 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5371 \dim_ratio:nn
5372 {
5373 \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
5374 + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5375 }
5376 { 2 \l_@@_l_dim }
5377 }
5378 \dim_gadd:Nn \l_@@_y_initial_dim
5379 {
5380 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5381 \dim_ratio:nn
5382 {
5383 \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
5384 + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5385 }
5386 { 2 \l_@@_l_dim }
5387 }
5388 \pgf@relevantforpicturesizefalse
5389 \int_step_inline:nnn \c_zero_int \l_1tmpa_int
5390 {
5391 \pgfpathcircle
5392 { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5393 \l_@@_xdots_radius_dim
5394 \dim_add:Nn \l_@@_x_initial_dim \l_1tmpa_dim
5395 \dim_add:Nn \l_@@_y_initial_dim \l_1tmpb_dim
5396 }
5397 \pgfusepathqfill
5398 }

```

```

5399 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
5400 {
5401   \pgfscope
5402   \pgftransformshift
5403   {
5404     \pgfpointlineattime { 0.5 }
5405     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5406     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
5407   }
5408   \fp_set:Nn \l_tmpa_fp
5409   {
5410     atand
5411     (
5412       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
5413       \l_@@_x_final_dim - \l_@@_x_initial_dim
5414     )
5415   }
5416   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
5417   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
5418   \tl_if_empty:NF \l_@@_xdots_middle_tl
5419   {
5420     \begin { pgfscope }
5421     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
5422     \pgfnode
5423     { rectangle }
5424     { center }
5425     {
5426       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5427       {
5428         $ % $
5429         \scriptstyle \l_@@_xdots_middle_tl
5430         $ % $
5431       }
5432     }
5433     { }
5434     {
5435       \pgfsetfillcolor { white }
5436       \pgfusepath { fill }
5437     }
5438     \end { pgfscope }
5439   }
5440   \tl_if_empty:NF \l_@@_xdots_up_tl
5441   {
5442     \pgfnode
5443     { rectangle }
5444     { south }
5445     {
5446       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5447       {
5448         $ % $
5449         \scriptstyle \l_@@_xdots_up_tl
5450         $ % $
5451       }
5452     }
5453     { }
5454     { \pgfusepath { } }
5455   }
5456   \tl_if_empty:NF \l_@@_xdots_down_tl
5457   {
5458     \pgfnode
5459     { rectangle }
5460     { north }
5461     {

```

```

5462         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5463         {
5464             $ % $
5465             \scriptstyle \l_@@_xdots_down_tl
5466             $ % $
5467         }
5468     }
5469     { }
5470     { \pgfusepath { } }
5471 }
5472 \endpgfscope
5473 }

```

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.

```

5474 \AtBeginDocument
5475 {

```

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

```

5476     \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
5477     \cs_new_protected:Npn \@@_Ldots: { \@@_collect_options:n { \@@_Ldots_i } }
5478     \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5479     {
5480         \int_if_zero:nTF \c@jCol
5481         { \@@_error:nn { in~first~col } { \Ldots } }
5482         {
5483             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5484             { \@@_error:nn { in~last~col } { \Ldots } }
5485             {
5486                 \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5487                 { #1 , down = #2 , up = #3 , middle = #4 }
5488             }
5489         }
5490     }
5491     \bool_if:NF \l_@@_nullify_dots_bool
5492     { \phantom { \ensuremath { \@@_old_ldots: } } }
5493     \bool_gset_true:N \g_@@_empty_cell_bool
5494 }

5494 \cs_new_protected:Npn \@@_Cdots: { \@@_collect_options:n { \@@_Cdots_i } }
5495 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5496 {
5497     \int_if_zero:nTF \c@jCol
5498     { \@@_error:nn { in~first~col } { \Cdots } }
5499     {
5500         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5501         { \@@_error:nn { in~last~col } { \Cdots } }
5502         {
5503             \@@_instruction_of_type:nnn { \c_false_bool } { \Cdots }

```

```

5504         { #1 , down = #2 , up = #3 , middle = #4 }
5505     }
5506 }
5507 \bool_if:NF \l_@@_nullify_dots_bool
5508 { \phantom { \ensuremath { \@@_old_cdots: } } }
5509 \bool_gset_true:N \g_@@_empty_cell_bool
5510 }

5511 \cs_new_protected:Npn \@@_Vdots: { \@@_collect_options:n { \@@_Vdots_i } }
5512 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5513 {
5514     \int_if_zero:nTF \c@iRow
5515     { \@@_error:nn { in~first~row } { \Vdots } }
5516     {
5517         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5518         { \@@_error:nn { in~last~row } { \Vdots } }
5519         {
5520             \@@_instruction_of_type:nnn { \c_false_bool } { \Vdots }
5521             { #1 , down = #2 , up = #3 , middle = #4 }
5522         }
5523     }
5524     \bool_if:NF \l_@@_nullify_dots_bool
5525     { \phantom { \ensuremath { \@@_old_vdots: } } }
5526     \bool_gset_true:N \g_@@_empty_cell_bool
5527 }

5528 \cs_new_protected:Npn \@@_Ddots: { \@@_collect_options:n { \@@_Ddots_i } }
5529 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5530 {
5531     \int_case:nnF \c@iRow
5532     {
5533         0 { \@@_error:nn { in~first~row } { \Ddots } }
5534         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5535     }
5536     {
5537         \int_case:nnF \c@jCol
5538         {
5539             0 { \@@_error:nn { in~first~col } { \Ddots } }
5540             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5541         }
5542         {
5543             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5544             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
5545             { #1 , down = #2 , up = #3 , middle = #4 }
5546         }
5547     }
5548 }
5549 \bool_if:NF \l_@@_nullify_dots_bool
5550 { \phantom { \ensuremath { \@@_old_ddots: } } }
5551 \bool_gset_true:N \g_@@_empty_cell_bool
5552 }

5553 \cs_new_protected:Npn \@@_Iddots:
5554 { \@@_collect_options:n { \@@_Iddots_i } }
5555 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5556 {
5557     \int_case:nnF \c@iRow
5558     {
5559         0 { \@@_error:nn { in~first~row } { \Iddots } }
5560         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5561     }

```

```

5562     {
5563         \int_case:nnF \c@jCol
5564         {
5565             0 { \@@_error:nn { in~first~col } { \Iddots } }
5566             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5567         }
5568         {
5569             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5570             \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { Iddots }
5571             { #1 , down = #2 , up = #3 , middle = #4 }
5572         }
5573     }
5574     \bool_if:NF \l_@@_nullify_dots_bool
5575     { \phantom { \ensuremath { \@@_old_iddots: } } }
5576     \bool_gset_true:N \g_@@_empty_cell_bool
5577 }
5578 }

```

End of the `\AddToHook`.

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

```

5579 \keys_define:nn { nicematrix / Ddots }
5580 {
5581     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5582     draw-first .value_forbidden:n = true ,
5583 }

```

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

```

5584 \cs_new_protected:Npn \@@_Hspace:
5585 {
5586     \bool_gset_true:N \g_@@_empty_cell_bool
5587     \hspace
5588 }

```

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`.

```

5589 \cs_new:Npn \@@_Hdotsfor:
5590 {
5591     \bool_lazy_and:nnTF
5592     { \int_if_zero_p:n \c@jCol }
5593     { \int_if_zero_p:n \l_@@_first_col_int }
5594     {
5595         \bool_if:NTF \g_@@_after_col_zero_bool
5596         {
5597             \multicolumn { 1 } { c } { }
5598             \@@_Hdotsfor_i:
5599         }
5600         { \@@_fatal:n { Hdotsfor~in~col~0 } }
5601     }
5602     {
5603         \multicolumn { 1 } { c } { }
5604         \@@_Hdotsfor_i:
5605     }
5606 }

```

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:`).

```

5607 \AtBeginDocument
5608 {

```

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

```
5609 \cs_new_protected:Npn \@@_Hdotsfor_i:
5610 { \@@_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).

```
5611 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5612 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5613 {
5614   \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5615   {
5616     \@@_Hdotsfor:nnnn
5617     { \int_use:N \c_iRow }
5618     { \int_use:N \c_jCol }
5619     { #2 }
5620     {
5621       #1 , #3 ,
5622       down = \exp_not:n { #4 } ,
5623       up = \exp_not:n { #5 } ,
5624       middle = \exp_not:n { #6 }
5625     }
5626   }
5627   \prg_replicate:nn { #2 - 1 }
5628   {
5629     &
5630     \multicolumn { 1 } { c } { }
5631     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5632   }
5633 }
5634 }
```

```
5635 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5636 {
5637   \bool_set_false:N \l_@@_initial_open_bool
5638   \bool_set_false:N \l_@@_final_open_bool
```

For the row, it's easy.

```
5639 \int_set:Nn \l_@@_initial_i_int { #1 }
5640 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
```

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

```
5641 \int_compare:nNnTF { #2 } = 1
5642 {
5643   \int_set:Nn \l_@@_initial_j_int 1
5644   \bool_set_true:N \l_@@_initial_open_bool
5645 }
5646 {
5647   \cs_if_exist:cTF
5648   {
5649     pgf @ sh @ ns @ \@@_env:
5650     - \int_use:N \l_@@_initial_i_int
5651     - \int_eval:n { #2 - 1 }
5652   }
5653   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5654   {
5655     \int_set:Nn \l_@@_initial_j_int { #2 }
5656     \bool_set_true:N \l_@@_initial_open_bool
5657   }
5658 }
5659 \int_compare:nNnTF { #2 + #3 - 1 } = \c_jCol
5660 {
5661   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5662   \bool_set_true:N \l_@@_final_open_bool
```

```

5663 }
5664 {
5665   \cs_if_exist:cTF
5666   {
5667     pgf @ sh @ ns @ \@@_env:
5668     - \int_use:N \l_@@_final_i_int
5669     - \int_eval:n { #2 + #3 }
5670   }
5671   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5672   {
5673     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5674     \bool_set_true:N \l_@@_final_open_bool
5675   }
5676 }
5677 \bool_if:NT \g_@@_aux_found_bool
5678 {
5679   {
5680     \@@_open_shorten:
5681     \int_if_zero:nTF { #1 }
5682     { \color { nicematrix-first-row } }
5683     {
5684       \int_compare:nNt { #1 } = \g_@@_row_total_int
5685       { \color { nicematrix-last-row } }
5686     }
5687     \keys_set:nn { nicematrix / xdots } { #4 }
5688     \@@_color:o \l_@@_xdots_color_tl
5689     \@@_actually_draw_Ldots:
5690   }
5691 }

```

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).

```

5692   \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5693   { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5694 }

```

```

5695 \AtBeginDocument
5696 {
5697   \cs_new_protected:Npn \@@_Vdotsfor:
5698   { \@@_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`).

```

5699   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5700   \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5701   {
5702     \bool_gset_true:N \g_@@_empty_cell_bool
5703     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5704     {
5705       \@@_Vdotsfor:nnnn
5706       { \int_use:N \c@iRow }
5707       { \int_use:N \c@jCol }
5708       { #2 }
5709       {
5710         #1 , #3 ,
5711         down = \exp_not:n { #4 } ,
5712         up = \exp_not:n { #5 } ,
5713         middle = \exp_not:n { #6 }
5714       }
5715     }
5716   }
5717 }

```

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

```

5718 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5719 {
5720   \bool_set_false:N \l_@@_initial_open_bool
5721   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5722   \int_set:Nn \l_@@_initial_j_int { #2 }
5723   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5724   \int_compare:nNnTF { #1 } = 1
5725   {
5726     \int_set:Nn \l_@@_initial_i_int 1
5727     \bool_set_true:N \l_@@_initial_open_bool
5728   }
5729   {
5730     \cs_if_exist:cTF
5731     {
5732       pgf @ sh @ ns @ \@@_env:
5733       - \int_eval:n { #1 - 1 }
5734       - \int_use:N \l_@@_initial_j_int
5735     }
5736     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5737     {
5738       \int_set:Nn \l_@@_initial_i_int { #1 }
5739       \bool_set_true:N \l_@@_initial_open_bool
5740     }
5741   }
5742   \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5743   {
5744     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5745     \bool_set_true:N \l_@@_final_open_bool
5746   }
5747   {
5748     \cs_if_exist:cTF
5749     {
5750       pgf @ sh @ ns @ \@@_env:
5751       - \int_eval:n { #1 + #3 }
5752       - \int_use:N \l_@@_final_j_int
5753     }
5754     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5755     {
5756       \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5757       \bool_set_true:N \l_@@_final_open_bool
5758     }
5759   }
5760   \bool_if:NT \g_@@_aux_found_bool
5761   {
5762     {
5763       \@@_open_shorten:
5764       \int_if_zero:nTF { #2 }
5765       { \color { nicematrix-first-col } }
5766       {
5767         \int_compare:nNnT { #2 } = \g_@@_col_total_int
5768         { \color { nicematrix-last-col } }
5769       }
5770       \keys_set:nn { nicematrix / xdots } { #4 }
5771       \@@_color:o \l_@@_xdots_color_tl
5772       \bool_if:NTF \l_@@_Vbrace_bool
5773       \@@_actually_draw_Vbrace:

```

```

5774         \@@_actually_draw_Vdots:
5775     }
5776 }

```

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).

```

5777     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5778     { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5779 }

```

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

```

5780 \NewDocumentCommand \@@_rotate: { 0 { } }
5781 {
5782     \bool_gset_true:N \g_@@_rotate_bool
5783     \keys_set:nn { nicematrix / rotate } { #1 }
5784     \ignorespaces
5785 }

```

The command `\@@_rotate_p_col:` will be linked to `\rotate` in the the cells of the columns of type *p* and *al*.

```

5786 \cs_new_protected:Npn \@@_rotate_p_col: { \@@_error:n { rotate-in-p-col } }

5787 \keys_define:nn { nicematrix / rotate }
5788 {
5789     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5790     c .value_forbidden:n = true ,
5791     -90 .code:n = \bool_gset_true:N \g_@@_rotate_minus_bool ,
5792     -90 .value_forbidden:n = true ,
5793     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5794 }

```

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).¹⁵

```

5795 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5796 {
5797     \tl_if_empty:nTF { #2 }
5798     { #1 }
5799     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5800 }
5801 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5802 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

¹⁵Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

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

```
5803 \AtBeginDocument
5804 {
```

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

```
5805   \tl_set_rescan:Nnn \l_tmpa_tl { }
5806   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5807   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5808   {
5809     {
5810       \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5811       \@@_color:o \l_@@_xdots_color_tl
5812       \use:e
5813       {
5814         \@@_line_i:nn
5815         { \@@_double_int_eval:n #2 - \q_stop }
5816         { \@@_double_int_eval:n #3 - \q_stop }
5817       }
5818     }
5819   }
5820 }

5821 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5822 {
5823   \bool_set_false:N \l_@@_initial_open_bool
5824   \bool_set_false:N \l_@@_final_open_bool
5825   \bool_lazy_or:nnTF
5826   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5827   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5828   { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }
```

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

```
5829   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5830 }

5831 \AtBeginDocument
5832 {
5833   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5834   {
```

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:`.

```
5835   \c_@@_pgfortikzpicture_tl
5836   \@@_draw_line_iii:nn { #1 } { #2 }
5837   \c_@@_endpgfortikzpicture_tl
5838 }
5839 }
```

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

```
5840 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5841 {
5842   \pgfrememberpicturepositiononpagetrue
5843   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5844   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5845   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5846   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5847   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5848   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5849   \@@_draw_line:
5850 }
```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don't use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

20 The command `\RowStyle`

`\g_@@_row_style_tl` may contain several instructions of the form:

```
@@_if_row_less_than:nn { number } { instructions }
```

Then, `\g_@@_row_style_tl` will be inserted in all the cells of the array (and also in both components of a `\diagbox` in a cell of in a mono-row block).

The test `@@_if_row_less_than:nn` ensures that the instructions are inserted only if you are in a row which is (still) in the scope of that instructions (which depends on the value of the key `nb-rows` of `\RowStyle`).

That test will be active even in an expandable context because `@@_if_row_less_than:nn` is *not* protected.

`#1` is the first row *after* the scope of the instructions in `#2`

However, both arguments are implicit because they are taken by curryfication.

```
5851 \cs_new:Npn @@_if_row_less_than:nn { \int_compare:nNt \c@iRow < }
5852 \cs_new:Npn @@_if_col_greater_than:nn { \int_compare:nNf \c@jCol < }
```

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

```
5853 \cs_set_protected:Npn @@_put_in_row_style:n #1
5854 {
5855   \tl_gput_right:Ne \g_@@_row_style_tl
5856   {
```

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

```
5857   \exp_not:N
5858   @@_if_row_less_than:nn
5859   { \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`).

```
5860   {
5861     \exp_not:N
5862     @@_if_col_greater_than:nn
5863     { \int_use:N \c@jCol }
5864     { \exp_not:n { #1 } \scan_stop: }
5865   }
5866 }
5867 }
5868 \cs_generate_variant:Nn @@_put_in_row_style:n { e }
```

```
5869 \keys_define:nn { nicematrix / RowStyle }
5870 {
5871   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5872   cell-space-top-limit+ .code:n =
5873     \dim_set:Nn \l_tmpa_dim { \l_@@_cell_space_top_limit_dim + #1 } ,
5874   cell-space-top-limit+ .value_required:n = true ,
5875   cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
5876   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5877   cell-space-bottom-limit+ .code:n =
5878     \dim_set:Nn \l_tmpb_dim { \l_@@_cell_space_bottom_limit_dim + #1 } ,
5879   cell-space-bottom-limit+ .value_required:n = true ,
5880   cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
5881   cell-space-limits .meta:n =
5882   {
5883     cell-space-top-limit = #1 ,
```

```

5884     cell-space-bottom-limit = #1 ,
5885   } ,
5886   cell-space-limits+ .meta:n =
5887   {
5888     cell-space-top-limit += #1 ,
5889     cell-space-bottom-limit += #1 ,
5890   } ,
5891   cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
5892   color .tl_set:N = \l_@@_color_tl ,
5893   color .value_required:n = true ,
5894   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5895   nb-rows .code:n =
5896     \str_if_eq:eeTF { #1 } { * }
5897     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5898     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5899   nb-rows .value_required:n = true ,
5900   fill .tl_set:N = \l_@@_fill_tl ,
5901   fill .value_required:n = true ,

```

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

```

5902   opacity .tl_set:N = \l_@@_opacity_tl ,
5903   opacity .value_required:n = true ,
5904   rowcolor .tl_set:N = \l_@@_fill_tl ,
5905   rowcolor .value_required:n = true ,
5906   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5907   rounded-corners .default:n = 4 pt ,
5908   unknown .code:n =
5909     \@@_unknown_key:nn
5910     { nicematrix / RowStyle }
5911     { Unknown-key-for-RowStyle }
5912 }

```

```

5913 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5914 {
5915   \group_begin:
5916   \tl_clear:N \l_@@_fill_tl
5917   \tl_clear:N \l_@@_opacity_tl
5918   \tl_clear:N \l_@@_color_tl
5919   \int_set:Nn \l_@@_key_nb_rows_int 1
5920   \dim_zero:N \l_@@_rounded_corners_dim
5921   \dim_zero:N \l_tmpa_dim
5922   \dim_zero:N \l_tmpb_dim
5923   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5924   \tl_if_empty:NF \l_@@_fill_tl
5925   {
5926     \@@_add_opacity_to_fill:
5927     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5928     {

```

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

```

5929     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5930     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5931     {
5932       \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5933       - *
5934     }
5935     { \dim_use:N \l_@@_rounded_corners_dim }
5936   }
5937 }
5938 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5939   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5940   {
5941     \@@_put_in_row_style:e
5942     {
5943       \@@_put_in_cell_after_hook:n
5944       {

```

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

```

5945         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5946         { \dim_use:N \l_tmpa_dim }
5947       }
5948     }
5949   }

```

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

```

5950   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5951   {
5952     \@@_put_in_row_style:e
5953     {
5954       \@@_put_in_cell_after_hook:n
5955       {
5956         \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5957         { \dim_use:N \l_tmpb_dim }
5958       }
5959     }
5960   }

```

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

```

5961   \tl_if_empty:NF \l_@@_color_tl
5962   {
5963     \@@_put_in_row_style:e
5964     {
5965       \mode_leave_vertical:
5966       \@@_color:n { \l_@@_color_tl }
5967     }
5968   }

```

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

```

5969   \bool_if:NT \l_@@_bold_row_style_bool
5970   {
5971     \@@_put_in_row_style:n
5972     {
5973       \exp_not:n
5974       {
5975         \if_mode_math:
5976           $ % $
5977           \bfseries \boldmath
5978           $ % $
5979         \else:
5980           \bfseries \boldmath
5981         \fi:
5982       }
5983     }
5984   }
5985   \group_end:
5986   \g_@@_row_style_tl
5987   \ignorespaces
5988 }

```

The following commande must *not* be protected.

```

5989 \cs_new:Npn \@@_rounded_from_row:n #1
5990 {
5991   \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

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

```

5992     { \int_eval:n { #1 } - 1 }
5993     {
5994         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5995         - \exp_not:n { \int_use:N \c@jCol }
5996     }
5997     { \dim_use:N \l_@@_rounded_corners_dim }
5998 }

```

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).

```

5999 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
6000 {

```

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.

```

6001     \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`.

```

6002     \str_if_in:nnF { #1 } { !! }
6003     {
6004         \seq_map_indexed_inline:Nn \g_@@_colors_seq

```

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

```

6005         { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
6006     }
6007     \int_if_zero:nTF \l_tmpa_int

```

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

```

6008     {
6009         \seq_gput_right:Nn \g_@@_colors_seq { #1 }
6010         \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
6011     }

```

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

```

6012     { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
6013 }
6014 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }

```

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

```

6015 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
6016 {
6017   \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
6018   {

```

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

```

6019   \group_begin:
6020   \pgfsetcornersarced
6021   { \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.

```

6022   \bool_if:NTF \l_@@_hvlines_bool
6023   {
6024     \pgfpathrectanglecorners
6025     {
6026       \pgfpointadd
6027       { \@@_qpoint:n { row-1 } }
6028       { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
6029     }
6030     {
6031       \pgfpointadd
6032       {
6033         \@@_qpoint:n
6034         { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
6035       }
6036       { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
6037     }
6038   }
6039   {
6040     \pgfpathrectanglecorners
6041     { \@@_qpoint:n { row-1 } }
6042     {
6043       \pgfpointadd
6044       {
6045         \@@_qpoint:n
6046         { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
6047       }
6048       { \pgfpoint \c_zero_dim \arrayrulewidth }
6049     }
6050   }
6051   \pgfusepath { clip }
6052   \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

6053   }
6054 }

```

The command `\@@_actually_color:` will actually fill the rectangles, color by color, as specified in the sequence `\g_@@_colors_seq`.

```

6055 \cs_new_protected:Npn \@@_actually_color:
6056 {
6057   \pgfpicture
6058   \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.

```

6059   \@@_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`).

```

6060 \seq_map_indexed_inline:Nn \g_@@_colors_seq
6061 {
6062   \int_compare:nNnTF { ##1 } = 1
6063   {
6064     \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
6065     \use:c { g_@@_color _ 1 _tl }
6066     \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
6067   }
6068   {
6069     \pgfscope
6070     \@@_color_opacity: ##2
6071     \use:c { g_@@_color _ ##1 _tl }
6072     \tl_gclear:c { g_@@_color _ ##1 _tl }
6073     \pgfusepath { fill }
6074     \endpgfscope
6075   }
6076 }
6077 \endpgfpicture
6078 }

```

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

```

6079 \cs_new_protected:Npn \@@_color_opacity:
6080 {
6081   \peek_meaning:NTF [
6082     \@@_color_opacity:w
6083     { \@@_color_opacity:w [ ] }
6084   }

```

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.

```

6085 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
6086 {
6087   \tl_clear:N \l_tmpa_tl
6088   \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.

```

6089   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
6090   \tl_if_empty:NTF \l_tmpb_tl
6091     \@declaredcolor
6092     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
6093 }

```

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

```

6094 \keys_define:nn { nicematrix / color-opacity }
6095 {
6096   opacity .tl_set:N          = \l_tmpa_tl ,
6097   opacity .value_required:n = true
6098 }

```

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

```

6099 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
6100 {
6101   \def \l_@@_rows_tl { #1 }
6102   \def \l_@@_cols_tl { #2 }
6103   \@@_cartesian_path:
6104 }

```

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

```

6105 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
6106 {
6107   \tl_if_blank:nF { #2 }
6108   {
6109     \@@_add_to_colors_seq:en
6110     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
6111     { \@@_cartesian_color:nn { #3 } { - } }
6112   }
6113 }

```

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

```

6114 \NewDocumentCommand \@@_columncolor { 0 { } m m }
6115 {
6116   \tl_if_blank:nF { #2 }
6117   {
6118     \@@_add_to_colors_seq:en
6119     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
6120     { \@@_cartesian_color:nn { - } { #3 } }
6121   }
6122 }

```

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

```

6123 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
6124 {
6125   \tl_if_blank:nF { #2 }
6126   {
6127     \@@_add_to_colors_seq:en
6128     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
6129     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
6130   }
6131 }

```

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

```

6132 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
6133 {
6134   \tl_if_blank:nF { #2 }
6135   {
6136     \@@_add_to_colors_seq:en
6137     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
6138     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
6139   }
6140 }

```

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

```

6141 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
6142 {
6143   \@@_cut_on_hyphen:w #1 \q_stop
6144   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6145   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6146   \@@_cut_on_hyphen:w #2 \q_stop
6147   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
6148   \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`.

```

6149 \@@_cartesian_path:n { #3 }
6150 }

```

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

```

6151 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
6152 {
6153   \clist_map_inline:nn { #3 }
6154   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
6155 }

6156 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
6157 {
6158   \int_step_inline:nn \c@iRow
6159   {
6160     \int_step_inline:nn \c@jCol
6161     {
6162       \int_if_even:nTF { ####1 + ##1 }
6163       { \@@_cellcolor [ #1 ] { #2 } }
6164       { \@@_cellcolor [ #1 ] { #3 } }
6165       { ##1 - ####1 }
6166     }
6167   }
6168 }

```

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”.

```

6169 \NewDocumentCommand \@@_arraycolor { 0 { } m }
6170 {
6171   \@@_rectanglecolor [ #1 ] { #2 }
6172   { 1 - 1 }
6173   { \int_use:N \c@iRow - \int_use:N \c@jCol }
6174 }

6175 \keys_define:nn { nicematrix / rowcolors }
6176 {
6177   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
6178   cols .tl_set:N = \l_@@_cols_tl ,
6179   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
6180   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
6181 }

```

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*.

```

6182 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
6183 {

```

`\l_@@_colors_seq` will be the list of colors.

```

6184   \seq_clear_new:N \l_@@_colors_seq
6185   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
6186   \tl_clear_new:N \l_@@_cols_tl
6187   \tl_set:Nn \l_@@_cols_tl { - }
6188   \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).

```

6189   \int_zero_new:N \l_@@_color_int
6190   \int_set:Nn \l_@@_color_int 1
6191   \bool_if:NT \l_@@_respect_blocks_bool
6192   {

```

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`).

```

6193     \seq_set_filter:NNn \l_tmpa_seq \g_@@_pos_of_blocks_seq
6194     { \@@_not_in_exterior_p:nnnnn ##1 }
6195 }

```

#2 is the list of intervals of rows.

```

6196     \clist_map_inline:nn { #2 }
6197     {
6198         \tl_set:Nn \l_tmpa_tl { ##1 }
6199         \tl_if_in:NnTF \l_tmpa_tl { - }
6200         { \@@_cut_on_hyphen:w ##1 \q_stop }
6201         { \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.

```

6202     \int_set:Nn \l_tmpa_int \l_tmpa_tl
6203     \int_set:Nn \l_@@_color_int
6204     { \bool_if:NTF \l_@@_rowcolors_restart_bool { 1 } \l_tmpa_tl }
6205     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
6206     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
6207     {

```

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

```

6208         \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).

```

6209         \bool_if:NT \l_@@_respect_blocks_bool
6210         {
6211             \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
6212             { \@@_intersect_our_row_p:nnnnn #####1 }
6213             \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

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

```

6214         }
6215         \tl_set:Nn \l_@@_rows_tl
6216         { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

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

```

6217         \tl_set:Nn \l_@@_color_tl
6218         {
6219             \@@_color_index:n
6220             {
6221                 \int_mod:nn
6222                 { \l_@@_color_int - 1 }
6223                 { \seq_count:N \l_@@_colors_seq }
6224                 + 1
6225             }
6226         }
6227         \tl_if_empty:NF \l_@@_color_tl
6228         {
6229             \use:e
6230             {
6231                 \@@_add_to_colors_seq:nn
6232                 { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
6233                 { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
6234             }
6235         }
6236         \int_incr:N \l_@@_color_int
6237         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
6238     }
6239 }
6240 }

```

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.

```
6241 \cs_new:Npn \@@_color_index:n #1
6242 {
```

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

```
6243   \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
6244   { \@@_color_index:n { #1 - 1 } }
6245   { \seq_item:Nn \l_@@_colors_seq { #1 } }
6246 }
```

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

```
6247 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
6248 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }
```

The braces around #3 and #4 are mandatory.

```
6249 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
6250 {
6251   \int_compare:nNtT { #3 } > \l_tmpb_int
6252   { \int_set:Nn \l_tmpb_int { #3 } }
6253 }
```

```
6254 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
6255 {
6256   \int_if_zero:nTF { #4 }
6257   \prg_return_false:
6258   {
6259     \int_compare:nNtTF { #2 } > \c@jCol
6260     \prg_return_false:
6261     \prg_return_true:
6262   }
6263 }
```

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

```
6264 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
6265 {
6266   \int_compare:nNtTF { #1 } > \l_tmpa_int
6267   \prg_return_false:
6268   {
6269     \int_compare:nNtTF \l_tmpa_int > { #3 }
6270     \prg_return_false:
6271     \prg_return_true:
6272   }
6273 }
```

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`).

```
6274 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
6275 {
6276   \dim_compare:nNtTF { #1 } = \c_zero_dim
6277   {
6278     \bool_if:NtF \l_@@_nocolor_used_bool
6279     { \@@_cartesian_path_normal_ii: }
6280     {
```

Even if the user has used the key `corners` the list of cells in the corners may be empty. That's why we test against `\l_@@_corners_cells_clist` and not `\l_@@_corners_clist`.

```

6281         \clist_if_empty:NTF \l_@@_corners_cells_clist
6282         { \@@_cartesian_path_normal_i:n { #1 } }
6283         { \@@_cartesian_path_normal_ii: }
6284     }
6285 }
6286 { \@@_cartesian_path_normal_i:n { #1 } }
6287 }

```

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.

```

6288 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
6289 {
6290     \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

6291     \clist_map_inline:Nn \l_@@_cols_tl
6292     {

```

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

```

6293         \def \l_tmpa_tl { ##1 }
6294         \tl_if_in:NnTF \l_tmpa_tl { - }
6295         { \@@_cut_on_hyphen:w ##1 \q_stop }
6296         { \def \l_tmpb_tl { ##1 } }
6297         \tl_if_empty:NTF \l_tmpa_tl
6298         { \def \l_tmpa_tl { 1 } }
6299         {
6300             \str_if_eq:eeT \l_tmpa_tl { * }
6301             { \def \l_tmpa_tl { 1 } }
6302         }
6303         \int_compare:nNnT \l_tmpa_tl > \g_@@_col_total_int
6304         { \@@_error:n { Invalid~col~number } }
6305         \tl_if_empty:NTF \l_tmpb_tl
6306         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6307         {
6308             \str_if_eq:eeT \l_tmpb_tl { * }
6309             { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6310         }
6311         \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
6312         { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

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

```

6313         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6314         \@@_qpoint:n { col - \l_tmpa_tl }
6315         \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
6316         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6317         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6318         \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
6319         \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.

```

6320     \clist_map_inline:Nn \l_@@_rows_tl
6321     {
6322         \def \l_tmpa_tl { #####1 }
6323         \tl_if_in:NnTF \l_tmpa_tl { - }
6324         { \@@_cut_on_hyphen:w #####1 \q_stop }
6325         { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
6326         \tl_if_empty:NTF \l_tmpa_tl
6327         { \def \l_tmpa_tl { 1 } }
6328         {
6329             \str_if_eq:eeT \l_tmpa_tl { * }
6330             { \def \l_tmpa_tl { 1 } }
6331         }

```

```

6332 \tl_if_empty:NTF \l_tmpb_tl
6333 { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6334 {
6335   \str_if_eq:eeT \l_tmpb_tl { * }
6336   { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6337 }
6338 \int_compare:nNnT \l_tmpa_tl > \g_@@_row_total_int
6339 { \@@_error:n { Invalid~row~number } }
6340 \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
6341 { \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`.

```

6342 \cs_if_exist:cF
6343 { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
6344 {
6345   \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
6346   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6347   \@@_qpoint:n { row - \l_tmpa_tl }
6348   \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6349   \pgfpathrectanglecorners
6350   { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6351   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6352 }
6353 }
6354 }
6355 }

```

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

```

6356 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
6357 {
6358   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6359   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6360 \clist_map_inline:Nn \l_@@_cols_tl
6361 {
6362   \@@_qpoint:n { col - ##1 }
6363   \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
6364   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6365   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6366   \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
6367   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

6368 \clist_map_inline:Nn \l_@@_rows_tl
6369 {
6370   \@@_if_in_corner:nF { #####1 - ##1 }
6371   {
6372     \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
6373     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6374     \@@_qpoint:n { row - #####1 }
6375     \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6376     \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
6377     {
6378       \pgfpathrectanglecorners
6379       { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6380       { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6381     }
6382   }
6383 }
6384 }
6385 }

```

The following command corresponds to a radius of the corners equal to 0 pt. This command is used by the commands `\@@_rowcolors`, `\@@_columncolor`, etc.

```
6386 \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

```
6387 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
6388 {
6389   \bool_set_true:N \l_@@_nocolor_used_bool
6390   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6391   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
```

We begin the loop over the columns.

```
6392 \clist_map_inline:Nn \l_@@_rows_tl
6393 {
6394   \clist_map_inline:Nn \l_@@_cols_tl
6395     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - ####1 } { } }
6396 }
6397 }
```

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`.

```
6398 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
6399 {
6400   \clist_set_eq:NN \l_tmpa_clist #1
6401   \clist_clear:N #1
6402   \clist_map_inline:Nn \l_tmpa_clist
6403     {
```

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

```
6404   \def \l_tmpa_tl { ##1 }
6405   \tl_if_in:NnTF \l_tmpa_tl { - }
6406     { \@@_cut_on_hyphen:w ##1 \q_stop }
6407     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
6408   \bool_lazy_or:nnT
6409     { \str_if_eq_p:ee \l_tmpa_tl { * } }
6410     { \tl_if_blank_p:o \l_tmpa_tl }
6411     { \def \l_tmpa_tl { 1 } }
6412   \bool_lazy_or:nnT
6413     { \str_if_eq_p:ee \l_tmpb_tl { * } }
6414     { \tl_if_blank_p:o \l_tmpb_tl }
6415     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6416   \int_compare:nNnT \l_tmpb_tl > { #2 }
6417     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6418   \int_step_tokens:nnn \l_tmpa_tl \l_tmpb_tl { \clist_put_right:Nn #1 }
6419 }
6420 }
```

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

```
6421 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
6422 {
6423   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6424   {
```

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`).

```
6425   \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
6426   { \int_use:N \c@iRow - \int_use:N \c@jCol }
6427 }
6428 \ignorespaces
6429 }
```

```

6430 \NewDocumentCommand \@@_cellcolor_error { 0 { } m }
6431 { \@@_error:n { cellcolor~in~Block } }
6432 % \end{macrocode}
6433 %
6434 % \begin{macrocode}
6435 \NewDocumentCommand \@@_rowcolor_error { 0 { } m }
6436 { \@@_error:n { rowcolor~in~Block } }
6437 % \end{macrocode}
6438 %
6439 % \bigskip
6440 % The following command will be linked to |\rowcolor| in the tabular.
6441 % \begin{macrocode}
6442 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
6443 {
6444   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6445   {
6446     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
6447     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6448     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
6449   }
6450   \ignorespaces
6451 }

```

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

```

6452 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
6453 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around #2 and #3 are mandatory.

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

```

6454 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
6455 {

```

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`.

```

6456   \seq_gclear:N \g_tmpa_seq
6457   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6458   { \@@_rowlistcolors_tabular:nnnn #1 }
6459   \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`.

```

6460   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
6461   {
6462     { \int_use:N \c@iRow }
6463     { \exp_not:n { #1 } }
6464     { \exp_not:n { #2 } }
6465     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
6466   }
6467   \ignorespaces
6468 }

```

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`).

```
6469 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
6470 {
6471   \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).

```
6472   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
6473   {
6474     \tl_gput_right:Ne \g_@@_pre_code_before_tl
6475     {
6476       \@@_rowlistcolors
6477       [ \exp_not:n { #2 } ]
6478       { #1 - \int_eval:n { \c@iRow - 1 } }
6479       { \exp_not:n { #3 } }
6480       [ \exp_not:n { #4 } ]
6481     }
6482   }
6483 }
```

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.

```
6484 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
6485 {
6486   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6487   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
6488   \seq_gclear:N \g_@@_rowlistcolors_seq
6489 }

6490 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6491 {
6492   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6493   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6494 }
```

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.

```
6495 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
6496 {
```

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").

```
6497   \seq_if_in:NVF \g_@@_columncolor_seq \c@jCol
6498   {
6499     \seq_gput_right:NV \g_@@_columncolor_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).

```
6500   \tl_gput_left:Ne \g_@@_pre_code_before_tl
6501   {
6502     \exp_not:N \columncolor [ #1 ]
6503     { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6504   }
6505 }
6506 }
```

```

6507 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6508 {
6509   \clist_map_inline:nn { #1 }
6510   {
6511     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6512     { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99!
6513     \columncolor { nocolor } { ##1 }
6514   }
6515 }
6516 \cs_new_protected:Npn \@@_EmptyRow:n #1
6517 {
6518   \clist_map_inline:nn { #1 }
6519   {
6520     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6521     { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99!
6522     \rowcolor { nocolor } { ##1 }
6523   }
6524 }

```

The following command must *not* be protected.

```

6525 \cs_new:Npn \@@_FalseRow
6526 {
6527   \noalign
6528   {
6529     \skip_vertical:n
6530     { - \box_ht_plus_dp:N \@arstrutbox + \l_@@_width_of_false_dim }
6531   }
6532   \multicolumn { 1 } { @{} c @{} } { } % added 2026-08-21

```

We keep in memory a list of the `\FalseRow` because we use it with `create-blocks-in-col`.

```

6533   \clist_gput_right:Ne \g_@@_false_rows_clist { \arabic { iRow } }
6534   \\
6535 }
6536 \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`.

```

6537 \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`.

```

6538 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6539 {
6540   \int_if_zero:nTF \l_@@_first_col_int
6541   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6542   {
6543     \int_if_zero:nTF \c@jCol
6544     {

```

```

6545         \int_compare:nNnF \c@iRow = { -1 }
6546         {
6547             \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 }
6548             { #1 }
6549         }
6550     }
6551     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6552 }
6553 }

```

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.

```

6554 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6555 {
6556     \int_if_zero:nF \c@iRow
6557     {
6558         \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6559         {
6560             \int_compare:nNnT \c@jCol > \c_zero_int
6561             { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6562         }
6563     }
6564 }

```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to -2 or -1 (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

The following command will be used for `\Toprule`, `\BottomRule` and `\MidRule`.

```

6565 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6566 {
6567     \IfPackageLoadedTF { tikz }
6568     {
6569         \IfPackageLoadedTF { booktabs }
6570         { #2 }
6571         { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6572     }
6573     { \@@_error:nn { TopRule~without~tikz } { #1 } }
6574 }
6575 \NewExpandableDocumentCommand { \@@_TopRule } { }
6576 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
6577 \cs_new:Npn \@@_TopRule_i:
6578 {
6579     \noalign \bgroup
6580     \peek_meaning:NTF [
6581     { \@@_TopRule_ii: }
6582     { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6583 }
6584 \NewDocumentCommand \@@_TopRule_ii: { o }
6585 {
6586     \tl_gput_right:Ne \g_@@_rules_tl
6587     {
6588         \@@_draw_hrulerule:n
6589         {
6590             position = \int_eval:n { \c@iRow + 1 } ,
6591             tikz =
6592             {
6593                 line~width = #1 ,
6594                 yshift = 0.25 \arrayrulewidth ,
6595                 shorten~< = - 0.5 \arrayrulewidth
6596             } ,

```

```

6597         total-width = #1
6598     }
6599 }
6600 \skip_vertical:n { \belowrulesep + #1 }
6601 \egroup
6602 }
6603 \NewExpandableDocumentCommand { \@@_BottomRule } { } {
6604   { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }
6605   \cs_new:Npn \@@_BottomRule_i:
6606   {
6607     \noalign \bgroup
6608     \peek_meaning:NTF [
6609       { \@@_BottomRule_ii: }
6610       { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6611     }
6612   \NewDocumentCommand \@@_BottomRule_ii: { o }
6613   {
6614     \tl_gput_right:Ne \g_@@_rules_tl
6615     {
6616       \@@_draw_hrulerule:n
6617       {
6618         position = \int_eval:n { \c@iRow + 1 } ,
6619         tikz =
6620         {
6621           line-width = #1 ,
6622           yshift = 0.25 \arrayrulewidth ,
6623           shorten~< = - 0.5 \arrayrulewidth
6624         } ,
6625         total-width = #1 ,
6626       }
6627     }
6628     \skip_vertical:N \aboverulesep
6629     \@@_create_row_node_i:
6630     \skip_vertical:n { #1 }
6631     \egroup
6632   }
6633 \NewExpandableDocumentCommand { \@@_MidRule } { } {
6634   { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6635   \cs_new:Npn \@@_MidRule_i:
6636   {
6637     \noalign \bgroup
6638     \peek_meaning:NTF [
6639       { \@@_MidRule_ii: }
6640       { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6641     }
6642   \NewDocumentCommand \@@_MidRule_ii: { o }
6643   {
6644     \skip_vertical:N \aboverulesep
6645     \@@_create_row_node_i:
6646     \tl_gput_right:Ne \g_@@_rules_tl
6647     {
6648       \@@_draw_hrulerule:n
6649       {
6650         position = \int_eval:n { \c@iRow + 1 } ,
6651         tikz =
6652         {
6653           line-width = #1 ,
6654           yshift = 0.25 \arrayrulewidth ,
6655           shorten~< = - 0.5 \arrayrulewidth
6656         } ,
6657         total-width = #1 ,

```

```

6658     }
6659   }
6660   \skip_vertical:n { \belowrulesep + #1 }
6661   \egroup
6662 }

```

General system for drawing rules

```

6663 \AtBeginDocument
6664 {
6665   \cs_new_protected:Npe \@@_draw_rules:
6666   {
6667     \c_@@_pgfortikzpicture_tl
6668     \@@_draw_rules_i:
6669     \c_@@_endpgfortikzpicture_tl
6670   }
6671 }
6672 \cs_new_protected:Npn \@@_draw_rules_i:
6673 {
6674   \bool_lazy_all:nT
6675   {
6676     { \clist_if_empty_p:N \l_@@_corners_clist }
6677     { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
6678     { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
6679     { \seq_if_empty_p:N \g_@@_pos_of_stroken_blocks_seq }
6680     { ! \l_@@_fix_vertex_bool }
6681   }
6682   {
6683     \socket_assign_plug:nn { nicematrix / draw-hrule } { quick }
6684     \socket_assign_plug:nn { nicematrix / draw-vrule } { quick }
6685   }
6686   \pgfrememberpicturepositiononpagetrue
6687   \pgf@relevantforpicturesizefalse
6688   \pgfsetrectcap
6689   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6690   \CT@arc@
6691   \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_key_hlines:
6692   \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_key_vlines:
6693   \g_@@_rules_tl
6694 }

```

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”.

```

6695 \keys_define:nn { nicematrix / rules-after }
6696 {
6697   position .int_set:N = \l_@@_position_int ,
6698   start .int_set:N = \l_@@_start_int ,
6699   start .initial:n = 1 ,
6700   end .int_set:N = \l_@@_end_int ,
6701   multiplicity .code:n = \@@_set_multiplicity:n { #1 } ,
6702   dotted .code:n =
6703     \bool_set_true:N \l_@@_dotted_bool
6704     \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
6705     \socket_assign_plug:nn { nicematrix / hsegment } { dotted } ,
6706   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].

```

6707     color .code:n =
6708         \@@_set_CTarc:n { #1 }
6709         \CT@arc@
6710         \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6711     color .value_required:n = true ,
6712     sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6713     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
    }
}

6714     total-width .dim_set:N = \l_@@_rule_width_after_dim ,
6715     unknown .code:n =
6716         \@@_unknown_key:nn
6717             { nicematrix / rules-after }
6718             { Unknown-key-for-a-rule } ,
6719     tikz .value_required:n = true
6720 }

```

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.

```

6721 \AtBeginDocument
6722 {
6723     \IfPackageLoadedTF { tikz }
6724     {
6725         \keys_define:nn { nicematrix / rules-after }
6726         {
6727             tikz .code:n =
6728                 \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 }
6729                 \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
6730                 \socket_assign_plug:nn { nicematrix / hsegment } { tikz } ,
6731             dashed .meta:n =
6732                 {
6733                     tikz = nicematrix / dashed ,
6734                     total-width = \pgflinewidth
6735                 }
6736         }
6737     }
6738     {
6739         \keys_define:nn { nicematrix / rules-after }
6740         { tikz .code:n = \@@_error:n { tikz~without~tikz } }
6741     }
6742 }

6743 \cs_new_protected:Npn \@@_set_multiplicity:n #1
6744 {
6745     \int_set:Nn \l_@@_multiplicity_int { #1 }
6746     \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
6747     \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
6748 }

6749 \AddToHook { package / tikz / after }
6750 {

```

```

6751 \tikzset
6752 {
6753     nicematrix / dashed / .style =
6754     {
6755         dash-pattern = on~4~pt~off~2pt ,
6756         line-cap = butt
6757     }
6758 }
6759 \cs_if_exist:cT { tikz@library@decorations@loaded }
6760 { \tikzset { nicematrix / dashed / .append-style = dash-expand-off } }
6761 }

```

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.

```

6762 \cs_new_protected:Npn \@@_draw_vrule:n #1
6763 {

```

The group is for the options.

```

6764 {
6765     \int_set_eq:NN \l_@@_end_int \c@iRow
6766     \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).

```

6767     \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6768     { \socket_use:n { nicematrix / draw-vrule } }
6769 }
6770 }

```

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.

```

6771 \socket_new:nn { nicematrix / draw-vrule } { 0 }
6772 \socket_new_plug:nnn { nicematrix / draw-vrule } { general }
6773 {
6774     \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`.

```

6775     \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).

```

6776     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6777     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6778     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6779     \l_tmpa_tl
6780     {

```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```

6781 \@@_test_v:
6782 \bool_if:NTF \g_tmpa_bool
6783 {
6784 \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.

```

6785 { \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl }
6786 { \socket_use:nn { nicematrix / draw-at-odd-vertex-v } }
6787 }
6788 {
6789 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6790 {
6791 \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }

```

The socket `vsegment` is defined below with its four plugs.

```

6792 \socket_use:n { nicematrix / vsegment }
6793 \int_zero:N \l_@@_segment_start_int
6794 }
6795 }
6796 }
6797 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6798 {
6799 \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6800 \socket_use:n { nicematrix / vsegment }
6801 }
6802 \group_end:
6803 }
6804 \socket_assign_plug:nn { nicematrix / draw-vrule } { general }
6805 \socket_new_plug:nnn { nicematrix / draw-vrule } { quick }
6806 {
6807 \group_begin:
6808 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6809 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6810 \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
6811 \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6812 \socket_use:n { nicematrix / vsegment }
6813 \group_end:
6814 }

```

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.

```

6815 \cs_new_protected:Npn \@@_test_v:
6816 {
6817 \bool_gset_true:N \g_tmpa_bool
6818 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6819 \bool_if:NT \g_tmpa_bool
6820 {
6821 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6822 { \@@_test_vline_in_block:nnnnn ##1 }
6823 \bool_if:NT \g_tmpa_bool
6824 {
6825 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6826 { \@@_test_vline_in_block:nnnnn ##1 }
6827 \bool_if:NT \g_tmpa_bool
6828 {
6829 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6830 { \@@_test_vline_in_stroken_block:nnnn ##1 }
6831 }
6832 }

```

```

6833     }
6834 }

6835 \socket_new:nn { nicematrix / draw-at-odd-vertex-v } { 0 }
6836 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-v } { active }
6837 {
6838   {
6839     \int_compare:nNnTF \l_@@_position_int > \c@jCol
6840     { \bool_set_false:N \l_tmpa_bool }
6841     {
6842       \@@_test_h:
6843       \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6844     }
6845     \int_compare:nNnTF \l_@@_position_int = 1
6846     { \bool_gset_false:N \g_tmpa_bool }
6847     {
6848       \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl - 1 } }
6849       \@@_test_h:
6850     }
6851     \bool_gset:Nn \g_tmpa_bool
6852     { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6853   }
6854   \bool_if:NT \g_tmpa_bool
6855   {
6856     \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }
6857     \socket_use:n { nicematrix / vsegment }
6858     \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl
6859   }
6860 }

6861 \cs_new_protected:Npn \@@_test_in_corner_v:
6862 {
6863   \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6864   {
6865     \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6866     { \bool_set_false:N \g_tmpa_bool }
6867   }
6868   {
6869     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6870     {
6871       \int_compare:nNnTF \l_tmpb_tl = 1
6872       { \bool_set_false:N \g_tmpa_bool }
6873       {
6874         \@@_if_in_corner:nT
6875         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6876         { \bool_set_false:N \g_tmpa_bool }
6877       }
6878     }
6879   }
6880 }

```

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`.

```

6881 \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`.

```

6882 \socket_new_plug:nnn { nicematrix / vsegment } { standard }
6883 {
6884   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6885   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6886   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6887   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6888   \pgfusepathqstroke
6889 }
6890 \socket_assign_plug:nn { nicematrix / vsegment } { standard }
6891 \socket_new_plug:nnn { nicematrix / vsegment } { multiple }
6892 {
6893   \dim_add:Nn \l_@@_x_initial_dim
6894   {
6895     - 0.5 \l_@@_rule_width_after_dim
6896     +
6897     ( \arrayrulewidth * \l_@@_multiplicity_int
6898       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6899   }
6900   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6901   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6902   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6903   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6904   \bool_lazy_and:nnT
6905   { \cs_if_exist_p:N \CT@drsc@ }
6906   { ! \tl_if_blank_p:o \CT@drsc@ }
6907   {
6908     {
6909       \CT@drsc@
6910       \dim_add:Nn \l_@@_y_initial_dim { 0.5 \arrayrulewidth }
6911       \dim_sub:Nn \l_@@_y_final_dim { 0.5 \arrayrulewidth }
6912       \dim_set:Nn \l_@@_tmpd_dim
6913       {
6914         \l_@@_x_initial_dim - ( \doublerulesep + \arrayrulewidth )
6915         * ( \l_@@_multiplicity_int - 1 )
6916       }
6917       \pgfpathrectanglecorners
6918       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6919       { \pgfpoint \l_@@_tmpd_dim \l_@@_y_final_dim }
6920       \pgfusepath { fill }
6921     }
6922   }
6923   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6924   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6925   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6926   {
6927     \dim_sub:Nn \l_@@_x_initial_dim { \arrayrulewidth + \doublerulesep }
6928     \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6929     \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6930   }
6931   \pgfusepathqstroke
6932 }
6933 \socket_new_plug:nnn { nicematrix / vsegment } { dotted }
6934 {
6935   \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6936   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6937   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6938   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6939   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6940   \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`

```
6941 \@@_draw_line:
6942 }
```

```
6943 \socket_new_plug:nnn { nicematrix / vsegment } { tikz }
6944 {
6945 }
```

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).

```
6946 \tl_if_empty:NF \l_@@_rule_color_tl
6947 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6948 \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6949 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6950 \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6951 \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
```

The following line can't be short-cutted.

```
6952 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6953 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6954 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim ) --
6955 ( \l_@@_x_initial_dim , \l_@@_y_final_dim ) ;
6956 }
6957 }
```

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).

```
6958 \cs_new_protected:Npn \@@_draw_key_vlines:
6959 {
6960 {
6961 \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
6962 \int_set:Nn \l_@@_end_int \c@iRow
```

There is a curriification in the following code.

```
6963 \str_if_eq:eeTF \l_@@_vlines_clist { all }
6964 { \@@_lines_step_inline:n \c@jCol }
6965 { \clist_map_inline:Nn \l_@@_vlines_clist }
6966 {
6967 \int_set:Nn \l_@@_position_int { ##1 }
6968 \socket_use:n { nicematrix / draw-vrule }
6969 }
6970 }
6971 }
```

The following command is used in `\@@_draw_key_vlines:` and in `\@@_draw_key_hlines:`.

```
6972 \cs_new_protected:Npn \@@_lines_step_inline:n #1
6973 {
6974 \int_step_inline:nnn
6975 { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
```

```

6976     {
6977     #1
6978     \bool_lazy_or:nnF \g_@@_delims_bool \l_@@_except_borders_bool
6979     { + 1 }
6980     }
6981 }

```

The horizontal rules

`\@@_draw_hrrule: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.

```

6982 \cs_new_protected:Npn \@@_draw_hrrule:n #1
6983 {
6984   {
6985     \int_set_eq:NN \l_@@_end_int \c_jCol
6986     \keys_set_known:nn { nicematrix / rules-after } { #1 }
6987     \socket_use:nn { nicematrix / draw-hrule }
6988   }
6989 }

```

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.

```

6990 \socket_new:nn { nicematrix / draw-hrule } { 0 }
6991 \socket_new_plug:nnn { nicematrix / draw-hrule } { general }
6992 {
6993   \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`.

```

6994   \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).

```

6995   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6996   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6997   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6998   \l_tmpb_tl
6999   {

```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```

7000   \@@_test_h:
7001   \bool_if:NTF \g_tmpa_bool
7002   {
7003     \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.

```

7004     { \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl }

```

```

7005         { \socket_use:n { nicematrix / draw-at-odd-vertex-h } }
7006     }
7007     {
7008         \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
7009         {
7010             \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }

```

The socket `hsegment` is defined below with its four plugs.

```

7011         \socket_use:n { nicematrix / hsegment }
7012         \int_zero:N \l_@@_segment_start_int
7013     }
7014 }
7015 }
7016 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
7017 {
7018     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
7019     \socket_use:n { nicematrix / hsegment }
7020 }
7021 \group_end:
7022 }
7023 \socket_assign_plug:nn { nicematrix / draw-hrule } { general }
7024 \socket_new_plug:nnn { nicematrix / draw-hrule } { quick }
7025 {
7026     \group_begin:
7027     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
7028     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
7029     \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
7030     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
7031     \socket_use:n { nicematrix / hsegment }
7032     \group_end:
7033 }

```

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.

```

7034 \cs_new_protected:Npn \@@_test_h:
7035 {
7036     \bool_gset_true:N \g_tmpa_bool
7037     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
7038     \bool_if:NT \g_tmpa_bool
7039     {
7040         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7041         { \@@_test_hline_in_block:nnnnn ##1 }
7042         \bool_if:NT \g_tmpa_bool
7043         {
7044             \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
7045             { \@@_test_hline_in_block:nnnnn ##1 }
7046             \bool_if:NT \g_tmpa_bool
7047             {
7048                 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
7049                 { \@@_test_hline_in_stroken_block:nnnn ##1 }
7050             }
7051         }
7052     }
7053 }
7054 \socket_new:nn { nicematrix / draw-at-odd-vertex-h } { 0 }
7055 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-h } { active }
7056 {
7057     {

```

```

7058 \int_compare:nNnTF \l_@@_position_int > \c@iRow
7059 { \bool_set_false:N \l_tmpa_bool }
7060 {
7061   \@@_test_v:
7062   \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
7063 }
7064 \int_compare:nNnTF \l_@@_position_int = 1
7065 { \bool_gset_false:N \g_tmpa_bool }
7066 {
7067   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl - 1 } }
7068   \@@_test_v:
7069 }
7070 \bool_gset:Nn \g_tmpa_bool
7071 { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
7072 }
7073 \bool_if:NT \g_tmpa_bool
7074 {
7075   \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }
7076   \socket_use:n { nicematrix / hsegment }
7077   \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl
7078 }
7079 }

7080 \cs_new_protected:Npn \@@_test_in_corner_h:
7081 {
7082   \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
7083   {
7084     \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
7085     { \bool_set_false:N \g_tmpa_bool }
7086   }
7087   {
7088     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
7089     {
7090       \int_compare:nNnTF \l_tmpa_tl = 1
7091       { \bool_set_false:N \g_tmpa_bool }
7092       {
7093         \@@_if_in_corner:nT
7094         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
7095         { \bool_set_false:N \g_tmpa_bool }
7096       }
7097     }
7098   }
7099 }

```

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`.

```

7100 \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`.

```

7101 \socket_new_plug:nnn { nicematrix / hsegment } { standard }
7102 {
7103   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
7104   \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }

```

```

7105 \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
7106 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7107 \pgfusepathqstroke
7108 }
7109 \socket_assign_plug:nn { nicematrix / hsegment } { standard }
7110 \socket_new_plug:nnn { nicematrix / hsegment } { multiple }
7111 {
7112 \dim_add:Nn \l_@@_y_initial_dim
7113 {
7114 - 0.5 \l_@@_rule_width_after_dim
7115 +
7116 ( \arrayrulewidth * \l_@@_multiplicity_int
7117 + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
7118 }
7119 \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
7120 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
7121 \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
7122 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
7123 \bool_lazy_and:nnT
7124 { \cs_if_exist_p:N \CT@drsc@ }
7125 { ! \tl_if_blank_p:o \CT@drsc@ }
7126 {
7127 {
7128 \CT@drsc@
7129 \dim_set:Nn \l_@@_tmpd_dim
7130 {
7131 \l_@@_y_initial_dim - ( \doublerulesep + \arrayrulewidth )
7132 * ( \l_@@_multiplicity_int - 1 )
7133 }
7134 \pgfpathrectanglecorners
7135 { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
7136 { \pgfpoint \l_@@_x_final_dim \l_@@_tmpd_dim }
7137 \pgfusepathqfill
7138 }
7139 }
7140 \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
7141 \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
7142 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
7143 {
7144 \dim_sub:Nn \l_@@_y_initial_dim { \arrayrulewidth + \doublerulesep }
7145 \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
7146 \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
7147 }
7148 \pgfusepathqstroke
7149 }

```

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}

```

$$\begin{array}{|cccc|}
\hline
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4 \\
\hline
\end{array}$$

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

7150 \socket_new_plug:nnn { nicematrix / hsegment } { dotted }
7151 {
7152   \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
7153   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
7154   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
7155   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
7156   \int_compare:nNnT \l_@@_segment_start_int = 1
7157   {
7158     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
7159     \bool_if:NF \g_@@_delims_bool
7160     { \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.

```

7161   \tl_if_eq:NnF \g_@@_left_delim_tl (
7162     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
7163   )
7164   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
7165   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
7166   \int_compare:nNnT \l_@@_segment_end_int = \c@jCol
7167   {
7168     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
7169     \bool_if:NF \g_@@_delims_bool
7170     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
7171     \tl_if_eq:NnF \g_@@_right_delim_tl )
7172     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
7173   }

```

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`

```

7174   \@@_draw_line:
7175 }

7176 \socket_new_plug:nnn { nicematrix / hsegment } { tikz }
7177 {
7178   {

```

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).

```

7179   \tl_if_empty:NF \l_@@_rule_color_tl
7180   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
7181   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
7182   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
7183   \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }

```

```

7184 \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
7185 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
7186 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
7187 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
7188 -- ( \l_@@_x_final_dim , \l_@@_y_initial_dim ) ;
7189 }
7190 }

```

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).

```

7191 \cs_new_protected:Npn \@@_draw_key_hlines:
7192 {
7193 {
7194 \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
7195 \int_set:Nn \l_@@_end_int \c@jCol

```

There is a currying in the following code.

```

7196 \str_if_eq:eeTF \l_@@_hlines_clist { all }
7197 { \@@_lines_step_inline:n \c@iRow }
7198 { \clist_map_inline:Nn \l_@@_hlines_clist }
7199 {
7200 \int_set:Nn \l_@@_position_int { ##1 }
7201 \socket_use:n { nicematrix / draw-hrule }
7202 }
7203 }
7204 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

7205 \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.

```

7206 \cs_set:Npn \@@_Hline_i:n #1
7207 {
7208 \peek_remove_spaces:n
7209 {
7210 \peek_meaning:NTF \Hline
7211 { \@@_Hline_ii:nn { #1 + 1 } }
7212 { \@@_Hline_iii:n { #1 } }
7213 }
7214 }

7215 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }

7216 \cs_set:Npn \@@_Hline_iii:n #1
7217 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }

7218 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
7219 {
7220 \@@_compute_rule_width:n { multiplicity = #1 , #2 }
7221 \skip_vertical:N \l_@@_rule_width_before_dim
7222 \tl_gput_right:Ne \g_@@_rules_tl
7223 {

```

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:

```

7224     {
7225         \int_compare:nNt { #1 } > 1 { \@@_set_multiplicity:n { #1 } }
7226         \int_set:Nn \l_@@_position_int { \int_eval:n { \c@iRow + 1 } }
7227         \dim_set:Nn \l_@@_rule_width_after_dim
7228             { \dim_use:N \l_@@_rule_width_before_dim }
7229         \@@_draw_hrule:n { #2 }
7230     }
7231 }
7232 \egroup
7233 }
```

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`).

```

7234 \cs_new_protected:Npn \@@_custom_line:n #1
7235 {
7236     \str_clear:N \l_@@_letter_str
7237     \str_clear:N \l_@@_command_str
7238     \str_clear:N \l_@@_ccommand_str
7239     \tl_clear_new:N \l_@@_other_keys_tl
7240     \keys_set_known:nn { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
7241     \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
7242     {
7243         \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }
```

We delete the last character which is a space.

```

7244     \str_set:Ne \l_@@_command_str
7245         { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
7246     }
7247     \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
7248     {
7249         \str_set:Ne \l_@@_ccommand_str
7250             { \str_tail:N \l_@@_ccommand_str }
7251         \str_set:Ne \l_@@_ccommand_str
7252             { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
7253     }
```

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.

```

7254     \bool_lazy_all:nTF
7255     {
7256         { \str_if_empty_p:N \l_@@_letter_str }
7257         { \str_if_empty_p:N \l_@@_command_str }
7258         { \str_if_empty_p:N \l_@@_ccommand_str }
7259     }
7260     { \@@_error:n { No~letter~and~no~command } }
7261     { \@@_custom_line_i:o \l_@@_other_keys_tl }
7262 }

7263 \keys_define:nn { nicematrix / custom-line }
7264 {
7265     letter .str_set:N = \l_@@_letter_str ,
7266     letter .value_required:n = true ,
7267     command .str_set:N = \l_@@_command_str ,
7268     command .value_required:n = true ,
7269     ccommand .str_set:N = \l_@@_ccommand_str ,
7270     ccommand .value_required:n = true
7271 }
```

```

7272 \cs_new_protected:Npn \@@_custom_line_i:n #1
7273 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

7274 \bool_set_false:N \l_@@_tikz_rule_bool
7275 \bool_set_false:N \l_@@_dotted_rule_bool
7276 \bool_set_false:N \l_@@_color_bool
7277 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
7278 \bool_if:NT \l_@@_tikz_rule_bool
7279 {
7280   \IfPackageLoadedF { tikz }
7281     { \@@_error:n { tikz~in~custom-line-without~tikz } }
7282   \bool_if:NT \l_@@_color_bool
7283     { \@@_error:n { color~in~custom-line-with~tikz } }
7284 }
7285 \bool_if:NT \l_@@_dotted_rule_bool
7286 {
7287   \int_compare:nNnT \l_@@_multiplicity_int > 1
7288     { \@@_error:n { key~multiplicity~with~dotted } }
7289 }
7290 \str_if_empty:NF \l_@@_letter_str
7291 {
7292   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
7293     { \@@_error:n { Several~letters } }
7294     {
7295       \tl_if_in:NoTF
7296         \c_@@_forbidden_letters_str
7297         \l_@@_letter_str
7298         { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
7299     }

```

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.

```

7300           \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
7301           { \@@_v_custom_line:nn { #1 } }
7302         }
7303       }
7304     }
7305     \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
7306     \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
7307   }
7308   \cs_generate_variant:Nn \@@_custom_line_i:n { o }
7309   \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbFVX|()[]!@<> }
7310   \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbFVX|()[]!@<> }

```

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.

```

7311 \keys_define:nn { nicematrix / custom-line-bis }
7312 {
7313   multiplicity .int_set:N = \l_@@_multiplicity_int ,
7314   color .code:n = \bool_set_true:N \l_@@_color_bool ,
7315   color .value_required:n = true ,
7316   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
7317   tikz .value_required:n = true ,
7318   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7319   dotted .value_forbidden:n = true ,
7320   dashed .meta:n =
7321   {

```

```

7322         tikz = nicematrix / dashed ,
7323         total-width = \pgflinewidth
7324     } ,
7325     total-width .code:n = { } ,
7326     total-width .value_required:n = true ,
7327     sep-color .code:n = { } ,
7328     sep-color .value_required:n = true ,
7329     unknown .code:n =
7330         \@@_unknown_key:nn
7331         { nicematrix / custom-line-bis }
7332         { Unknown-key-for-custom-line }
7333 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

7334 \bool_new:N \l_@@_dotted_rule_bool
7335 \bool_new:N \l_@@_tikz_rule_bool
7336 \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).

```

7337 \keys_define:nn { nicematrix / custom-line-width }
7338 {
7339     multiplicity .int_set:N = \l_@@_tmpc_int ,
7340     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
7341     total-width .code:n = \dim_set:Nn \l_@@_rule_width_before_dim { #1 }
7342         \bool_set_true:N \l_@@_total_width_bool ,
7343     total-width .value_required:n = true ,
7344     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7345 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule (hence the ‘h’ in the name) with the full width of the array. `#1` is the whole set of keys to pass to the command `\@@_draw_hrule:n` (which is in the internal `\CodeAfter`).

```

7346 \cs_new_protected:Npn \@@_h_custom_line:n #1
7347 {

```

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`).

```

7348     \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
7349     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
7350 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule on only some of the columns of the array (hence the letter `c` as in `\cline`). `#1` is the whole set of keys to pass to the command `\@@_draw_hrule:n` (which is in the internal `\CodeAfter`).

```

7351 \cs_new_protected:Npn \@@_c_custom_line:n #1
7352 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

7353     \exp_args:Nc \DeclareExpandableDocumentCommand
7354     { nicematrix - \l_@@_ccommand_str }
7355     { 0 { } m }
7356     {
7357         \noalign
7358         {
7359             \@@_compute_rule_width:n { #1 , ##1 }
7360             \skip_vertical:n \l_@@_rule_width_before_dim
7361             \clist_map_inline:nn
7362             { ##2 }

```

```

7363         { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
7364     }
7365 }
7366 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
7367 }

```

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*.

```

7368 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
7369 {
7370     \tl_if_in:nnTF { #2 } { - }
7371     { \@@_cut_on_hyphen:w #2 \q_stop }
7372     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
7373     \tl_gput_right:Ne \g_@@_rules_tl
7374     {
7375         \@@_draw_hrule:n
7376         {
7377             #1 ,
7378             start = \l_tmpa_tl ,
7379             end = \l_tmpb_tl ,
7380             position = \int_eval:n { \c@iRow + 1 } ,
7381             total-width = \dim_use:N \l_@@_rule_width_before_dim
7382         }
7383     }
7384 }
7385 \cs_new_protected:Npn \@@_compute_rule_width:n #1
7386 {
7387     \bool_set_false:N \l_@@_tikz_rule_bool
7388     \bool_set_false:N \l_@@_total_width_bool
7389     \bool_set_false:N \l_@@_dotted_rule_bool
7390     \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
7391     \bool_if:NF \l_@@_total_width_bool
7392     {
7393         \bool_if:NTF \l_@@_dotted_rule_bool
7394         { \dim_set:Nn \l_@@_rule_width_before_dim { 2 \l_@@_xdots_radius_dim } }
7395         {
7396             \bool_if:NF \l_@@_tikz_rule_bool
7397             {
7398                 \dim_set:Nn \l_@@_rule_width_before_dim
7399                 {
7400                     \arrayrulewidth * \l_@@_tmpc_int
7401                     + \doublerulesep * ( \l_@@_tmpc_int - 1 )
7402                 }
7403             }
7404         }
7405     }
7406 }

```

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

```

7407 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
7408 {
7409     \str_if_eq:nnTF { #2 } { [ ] }
7410     { \@@_v_custom_line_i:nw { #1 } [ ] }
7411     { \@@_v_custom_line_ii:nn { #2 } { #1 } }
7412 }
7413 \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
7414 { \@@_v_custom_line:nn { #1 , #2 } }
7415 \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
7416 {
7417     \@@_compute_rule_width:n { #2 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

7418 \tl_put_right:Ne \l_@@_array_preamble_tl
7419 {
7420   \exp_not:N !
7421   { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_before_dim } }
7422 }
7423 \tl_gput_right:Ne \g_@@_rules_tl
7424 {

```

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.

```

7425 {
7426   \dim_set:Nn \l_@@_rule_width_after_dim
7427   { \dim_use:N \l_@@_rule_width_before_dim }

```

Here, the `\int_eval:n` is mandatory because we expand an instruction written on `\g_@@_rules_tl` (which will be executed later).

```

7428   \int_set:Nn \l_@@_position_int
7429   { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
7430   \@@_draw_vrule:n { #2 }
7431 }
7432 }
7433 \@@_rec_preamble:n #1
7434 }

7435 \@@_custom_line:n
7436 { letter = : , command = \hdottedline , ccommand = \cdottedline, dotted }

```

The key default-line

```

7437 \keys_define:nn { nicematrix / default-line }
7438 {
7439   multiplicity .int_set:N = \l_@@_multiplicity_int ,
7440   color .code:n = \bool_set_true:N \l_@@_color_bool ,
7441   color .value_required:n = true ,
7442   dashed .code:n = \@@_error:n { dashed-for-default-line-without-TikZ } ,
7443   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7444   dotted .value_forbidden:n = true ,
7445   total-width .code:n = { } ,
7446   total-width .value_required:n = true ,
7447   sep-color .code:n = { } ,
7448   sep-color .value_required:n = true ,
7449   unknown .code:n =
7450     \@@_unknown_key:nn
7451     { nicematrix / default-line }
7452     { Unknown-key-for-default-line }
7453 }

7454 \AddToHook { package / tikz / after }
7455 {
7456   \keys_define:nn { nicematrix / default-line }
7457   {
7458     tikz .code:n =
7459       \bool_set_true:N \l_@@_tikz_rule_bool
7460       \tl_set:Nn \l_@@_tikz_rule_tl { #1 } ,
7461     tikz .value_required:n = true ,
7462     dashed .meta:n =
7463     {

```

```

7464         tikz = nicematrix / dashed ,
7465         total-width = \pgflinewidth
7466     }
7467 }
7468 \@@_custom_line:n
7469 {
7470     letter = ; ,
7471     command = \hdashedline ,
7472     ccommand = \cdashedline ,
7473     tikz = nicematrix / dashed ,
7474     total-width = \pgflinewidth
7475 }
7476 }
7477 \cs_new_protected:Npn \@@_default_line:n #1
7478 {
7479     \bool_set_false:N \l_@@_tikz_rule_bool
7480     \bool_set_false:N \l_@@_dotted_rule_bool
7481     \bool_set_false:N \l_@@_color_bool
7482     \keys_set:nn { nicematrix / default-line } { #1 }
7483     \bool_if:NT \l_@@_tikz_rule_bool
7484     {
7485         \IfPackageLoadedF { tikz }
7486         { \@@_error:n { tikz~in~custom~line~without~tikz } }
7487         \bool_if:NT \l_@@_color_bool
7488         { \@@_error:n { color~in~custom~line~with~tikz } }
7489     }
7490     \bool_if:NT \l_@@_dotted_rule_bool
7491     {
7492         \int_compare:nNnT \l_@@_multiplicity_int > 1
7493         { \@@_error:n { key~multiplicity~with~dotted } }
7494     }
7495     \bool_if:NTF \l_@@_tikz_rule_bool
7496     {
7497         \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
7498         \socket_assign_plug:nn { nicematrix / hsegment } { tikz }
7499     }
7500     {
7501         \bool_if:NTF \l_@@_dotted_rule_bool
7502         {
7503             \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
7504             \socket_assign_plug:nn { nicematrix / hsegment } { dotted }
7505         }
7506         {
7507             \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
7508             \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
7509         }
7510     }
7511 }

```

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.

Here is a version with the standard syntax of L3.

```

\cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
{
    \int_compare:nNnT \l_tmpa_tl > { #1 }
    {
        \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
        {

```

```

\int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
{
  \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
  { \bool_gset_false:N \g_tmpa_bool }
}
}
}

```

We will use a version a little more efficient.

```

7512 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
7513 {
7514   \if_int_compare:w \l_tmpa_tl > #1
7515   \if_int_compare:w \l_tmpa_tl > #3 \else:
7516     \if_int_compare:w \l_tmpb_tl < #2 \else:
7517       \if_int_compare:w \l_tmpb_tl > #4 \else:
7518         \bool_gset_false:N \g_tmpa_bool
7519       \fi:
7520     \fi:
7521   \fi:
7522 \fi:
7523 }

```

The same for vertical rules.

Here is a version with the standard syntax of L3.

```

\cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
{
  \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
  {
    \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
    {
      \int_compare:nNnT \l_tmpb_tl > { #2 }
      {
        \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
        { \bool_gset_false:N \g_tmpa_bool }
      }
    }
  }
}

```

We will use a version a little more efficient.

```

7524 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
7525 {
7526   \if_int_compare:w \l_tmpa_tl < #1 \else:
7527   \if_int_compare:w \l_tmpa_tl > #3 \else:
7528     \if_int_compare:w \l_tmpb_tl > #2
7529     \if_int_compare:w \l_tmpb_tl > #4 \else:
7530       \bool_gset_false:N \g_tmpa_bool
7531     \fi:
7532   \fi:
7533 \fi:
7534 \fi:
7535 }

```

Now, for the stroken blocks.

```

7536 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
7537 {
7538   % \int_compare:nNnT \l_tmpb_tl < { #2 - 1 }
7539   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
7540   {

```

```

7541     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
7542     {
7543         \int_compare:nNnTF \l_tmpa_tl = { #1 }
7544         { \bool_gset_false:N \g_tmpa_bool }
7545         {
7546             \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
7547             { \bool_gset_false:N \g_tmpa_bool }
7548         }
7549     }
7550 }
7551 }

7552 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
7553 {
7554     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
7555     {
7556         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
7557         {
7558             \int_compare:nNnTF \l_tmpb_tl = { #2 }
7559             { \bool_gset_false:N \g_tmpa_bool }
7560             {
7561                 \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
7562                 { \bool_gset_false:N \g_tmpa_bool }
7563             }
7564         }
7565     }
7566 }

```

23 The empty corners

```

7567 \cs_new_protected:Npn \@@_mark_blocks:
7568 {
7569     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7570     { \@@_mark_cells_of_block:nnnnn ##1 }
7571
7572     \cs_set_eq:NN \@@_mark_blocks: \prg_do_nothing:
7573 }

```

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.

```

7574 \cs_new_protected:Npn \@@_compute_corners:
7575 {
7576     \@@_mark_blocks:

```

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. However, for efficiency, the cells in the corners will also be marked directly in the main hash table of TeX.

```

7577     \clist_clear:N \l_@@_corners_cells_clist
7578     \clist_map_inline:Nn \l_@@_corners_clist
7579     {
7580         \str_case:nnF { ##1 }
7581         {
7582             { NW }
7583             { \@@_compute_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
7584             { NE }
7585             { \@@_compute_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
7586             { SW }
7587             { \@@_compute_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
7588             { SE }

```

```

7589 { \@@_compute_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
7590 { N }
7591 { \@@_compute_corner_NS:nnnn \c@jCol 1 1 \c@iRow }
7592 { S }
7593 { \@@_compute_corner_NS:nnnn \c@jCol \c@iRow { -1 } 1 }
7594 { E }
7595 { \@@_compute_corner_EW:nnnn \c@iRow \c@jCol { -1 } 1 }
7596 { W }
7597 { \@@_compute_corner_EW:nnnn \c@iRow 1 1 \c@jCol }
7598 }
7599 { \@@_error:nn { bad~corner } { ##1 } }
7600 }

```

Even if the user has used the key corners, the list of cells in the corners may be empty. That's why we test against `\l_@@_corners_cells_clist` and not against `\l_@@_corners_clist`.

```

7601 \clist_if_empty:NF \l_@@_corners_cells_clist
7602 {

```

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.

```

7603 \tl_gput_right:Ne \g_@@_aux_tl
7604 {
7605   \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
7606   { \l_@@_corners_cells_clist }
7607 }
7608 }
7609 }

```

```

7610 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
7611 {
7612   \int_step_inline:nnn { #1 } { #3 }
7613   {
7614     \int_step_inline:nnn { #2 } { #4 }
7615     { \cs_set_nopar:cpn { @@ _ block _ ##1 - ####1 } { } }
7616   }
7617 }

```

```

7618 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p , TF , T , F }
7619 {
7620   \cs_if_exist:cTF
7621   { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
7622   \prg_return_true:
7623   \prg_return_false:
7624 }

```

When we have to “mark” an empty cell, we will:

- put it in `\l_@@_corners_cells_clist`; that `clist` will be written on the aux file because you have to know the empty cells during the `\CodeBefore`;
- mark directly the empty cell by an entry in the main hash table of TeX (for efficiency).

```

7625 \cs_new_protected:Npn \@@_mark_empty_cell:nn #1 #2
7626 {
7627   \clist_put_right:Ne \l_@@_corners_cells_clist { #1 - #2 }
7628   \cs_set_nopar:cpn { @@ _ corner _ #1 - #2 } { }
7629 }

```

“Computing a corner” is determining all the empty cells (which are not in a block) that belong to that corner.

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.

```
7630 \cs_new_protected:Npn \@@_compute_corner:nnnnnn #1 #2 #3 #4 #5 #6
7631 {
```

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.

```
7632   \int_zero_new:N \l_@@_last_empty_row_int
7633   \int_set:Nn \l_tmpa_int { #1 }
7634   \bool_set_false:N \l_tmpa_bool
7635   \bool_do_until:Nn \l_tmpa_bool
7636   {
7637     \bool_lazy_or:nnTF
7638     {
7639       \cs_if_exist_p:c
7640       {
7641         pgf @ sh @ ns @ \@@_env: -
7642         \int_use:N \l_tmpa_int - \int_eval:n { #2 }
7643       }
7644     }
7645     { \@@_if_in_block_p:nn \l_tmpa_int { #2 } }
7646     {
7647       \bool_set_true:N \l_tmpa_bool
7648       \int_compare:nNnTF \l_tmpa_int = { #1 }
7649       { \int_set_eq:NN \l_@@_last_empty_row_int \l_tmpa_int }
7650       { \int_set:Nn \l_@@_last_empty_row_int { \l_tmpa_int - #3 } }
7651     }
7652     {
7653       \int_compare:nNnTF \l_tmpa_int = { #5 }
7654       {
7655         \bool_set_true:N \l_tmpa_bool
7656         \int_set_eq:NN \l_@@_last_empty_row_int \l_tmpa_int
7657       }
7658       { \int_add:Nn \l_tmpa_int { #3 } }
7659     }
7660   }
```

Now, you determine the last empty cell in the row of number 1.

```
7661   \int_zero_new:N \l_@@_last_empty_column_int
7662   \int_set:Nn \l_tmpa_int { #2 }
7663   \bool_set_false:N \l_tmpa_bool
7664   \bool_do_until:Nn \l_tmpa_bool
7665   {
7666     \bool_lazy_or:nnTF
7667     {
7668       \cs_if_exist_p:c
7669       {
7670         pgf @ sh @ ns @ \@@_env: -
7671         \int_eval:n { #1 } - \int_use:N \l_tmpa_int
7672       }
7673     }
7674     { \@@_if_in_block_p:nn { #1 } \l_tmpa_int }
7675     {
7676       \bool_set_true:N \l_tmpa_bool
7677       \int_compare:nNnTF \l_tmpa_int = { #1 }
7678       { \int_set_eq:NN \l_@@_last_empty_column_int \l_tmpa_int }
7679       { \int_set:Nn \l_@@_last_empty_column_int { \l_tmpa_int - #4 } }
7680     }
7681   }
```

```

7681     {
7682         \int_compare:nNnTF \l_tmpa_int = { #6 }
7683         {
7684             \bool_set_true:N \l_tmpa_bool
7685             \int_set_eq:NN \l_@@_last_empty_column_int \l_tmpa_int
7686         }
7687         { \int_add:Nn \l_tmpa_int { #4 } }
7688     }
7689 }

```

Now, we loop over the rows.

```

7690     \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
7691     {

```

We treat the row number ##1 with a loop \bool_do_until:Nn.

```

7692         \int_set:Nn \l_tmpa_int { #2 }
7693         \bool_set_false:N \l_tmpa_bool
7694         \bool_do_until:Nn \l_tmpa_bool
7695         {
7696             \bool_lazy_or:nnTF
7697             {
7698                 \cs_if_exist_p:c
7699                 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_tmpa_int }
7700             }
7701             { \@@_if_in_block_p:nn { ##1 } \l_tmpa_int }
7702             { \bool_set_true:N \l_tmpa_bool }
7703             {
7704                 \@@_mark_empty_cell:nn { ##1 } { \int_use:N \l_tmpa_int }
7705                 \int_compare:nNnTF \l_tmpa_int = \l_@@_last_empty_column_int
7706                 { \bool_set_true:N \l_tmpa_bool }
7707                 \int_add:Nn \l_tmpa_int { #4 }
7708             }
7709         }
7710         \int_set:Nn \l_@@_last_empty_column_int { \l_tmpa_int - #4 }
7711     }
7712 }

```

For the “corners” N and S

```

7713 \cs_new_protected:Npn \@@_compute_corner_NS:nnnn #1 #2 #3 #4
7714 {
7715     \int_step_inline:nn { #1 }
7716     {
7717         \int_set:Nn \l_tmpa_int { #2 }
7718         \bool_set_false:N \l_tmpa_bool
7719         \bool_do_until:Nn \l_tmpa_bool
7720         {
7721             \bool_lazy_or:nnTF
7722             {
7723                 \cs_if_exist_p:c
7724                 { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_tmpa_int - ##1 }
7725             }
7726             { \@@_if_in_block_p:nn \l_tmpa_int { ##1 } }
7727             { \bool_set_true:N \l_tmpa_bool }
7728             {
7729                 \@@_mark_empty_cell:nn { \int_use:N \l_tmpa_int } { ##1 }
7730                 \int_compare:nNnTF \l_tmpa_int = { #4 }
7731                 { \bool_set_true:N \l_tmpa_bool }
7732                 { \int_add:Nn \l_tmpa_int { #3 } }
7733             }
7734         }
7735     }
7736 }

```

For the “corners” E and W.

```

7737 \cs_new_protected:Npn \@@_compute_corner_EW:nnnn #1 #2 #3 #4
7738 {
7739   \int_step_inline:nn { #1 }
7740   {
7741     \int_set:Nn \l_tmpa_int { #2 }
7742     \bool_set_false:N \l_tmpa_bool
7743     \bool_do_until:Nn \l_tmpa_bool
7744     {
7745       \bool_lazy_or:nnTF
7746       {
7747         \cs_if_exist_p:c
7748         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_tmpa_int }
7749       }
7750       { \@@_if_in_block_p:nn { ##1 } \l_tmpa_int }
7751       { \bool_set_true:N \l_tmpa_bool }
7752       {
7753         \@@_mark_empty_cell:nn { ##1 } { \int_use:N \l_tmpa_int }
7754         \int_compare:nNnTF \l_tmpa_int = { #4 }
7755         { \bool_set_true:N \l_tmpa_bool }
7756         { \int_add:Nn \l_tmpa_int { #3 } }
7757       }
7758     }
7759   }
7760 }

```

Of course, instead of the following lines, we could have used `\prg_new_conditional:Npnn`.

```

7761 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
7762 \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 far less efficient:

```
\clist_if_in:NnT \l_@@_corners_cells_clist { #1 } ...
```

24 The environment {NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```
7763 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

7764 \keys_define:nn { nicematrix / NiceMatrixBlock }
7765 {
7766   auto-columns-width .code:n =
7767   {
7768     \bool_set_true:N \l_@@_block_auto_columns_width_bool
7769     \dim_gzero_new:N \g_@@_max_cell_width_dim
7770     \bool_set_true:N \l_@@_auto_columns_width_bool
7771   }
7772 }

7773 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
7774 {
7775   \int_gincr:N \g_@@_NiceMatrixBlock_int
7776   \dim_zero:N \l_@@_columns_width_dim
7777   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
7778   \bool_if:NT \l_@@_block_auto_columns_width_bool

```

```

7779 {
7780   \cs_if_exist:cT
7781     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
7782     {
7783       \dim_set:Nn \l_@@_columns_width_dim
7784       {
7785         \use:c
7786           { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7787       }
7788     }
7789   }
7790 }

```

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`).

```

7791 {
7792   \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.

```

7793   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
7794   {
7795     \bool_if:NT \l_@@_block_auto_columns_width_bool
7796     {
7797       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
7798       \iow_shipout:Ne \@mainaux
7799       {
7800         \cs_gset:cpn
7801           { @@ _ 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.

```

7802         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7803       }
7804       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
7805     }
7806   }
7807   \ignorespacesafterend
7808 }

```

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).

```

7809 \cs_new_protected:Npn \@@_create_extra_nodes:
7810 {
7811   \bool_if:nTF \l_@@_medium_nodes_bool
7812   {
7813     \bool_if:NTF \l_@@_no_cell_nodes_bool
7814     { \@@_error:n { extra-nodes~with~no~cell~nodes } }
7815     {
7816       \bool_if:NTF \l_@@_large_nodes_bool
7817       \@@_create_medium_and_large_nodes:
7818       \@@_create_medium_nodes:
7819     }
7820   }
7821   {
7822     \bool_if:NT \l_@@_large_nodes_bool

```

```

7823     {
7824         \bool_if:NTF \l_@@_no_cell_nodes_bool
7825         { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7826         \@@_create_large_nodes:
7827     }
7828 }
7829 }

```

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`.

```

7830 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7831 {
7832     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7833     {
7834         \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7835         \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7836         \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7837         \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7838     }
7839     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7840     {
7841         \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7842         \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7843         \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7844         \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7845     }

```

We begin the two nested loops over the rows and the columns of the array.

```

7846     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7847     {
7848         \int_step_variable:nnNn
7849         \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.

```

7850     {
7851         \cs_if_exist:cT
7852         { 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`.

```

7853     {
7854         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7855         \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
7856         { \dim_min:vn { l_@@_row_ \@@_i: _min_dim } \pgf@y }
7857         \seq_if_in:Nef \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7858         {

```

```

7859         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7860         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7861     }

```

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`.

```

7862     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7863     \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
7864     { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
7865     \seq_if_in:Nef \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7866     {
7867         \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
7868         { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
7869     }
7870 }
7871 }
7872 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7873 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7874 {
7875     \dim_compare:nNnT
7876     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
7877     {
7878         \@@_qpoint:n { row - \@@_i: - base }
7879         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
7880         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
7881     }
7882 }
7883 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7884 {
7885     \dim_compare:nNnT
7886     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
7887     {
7888         \@@_qpoint:n { col - \@@_j: }
7889         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
7890         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
7891     }
7892 }
7893 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7894 \cs_new_protected:Npn \@@_create_medium_nodes:
7895 {
7896     \pgfpicture
7897     \pgfrememberpicturepositiononpagetrue
7898     \pgf@relevantforpicturesizefalse
7899     \@@_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”.

```

7900     \tl_set:Nn \l_@@_suffix_tl { -medium }
7901     \@@_create_nodes:
7902     \endpgfpicture
7903 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones¹⁶. However, the computation of the mathematical coordinates

¹⁶If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

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:`.

```

7904 \cs_new_protected:Npn \@@_create_large_nodes:
7905 {
7906   \pgfpicture
7907     \pgfrememberpicturepositiononpagetrue
7908     \pgf@relevantforpicturesizefalse
7909     \@@_computations_for_medium_nodes:
7910     \@@_computations_for_large_nodes:
7911     \tl_set:Nn \l_@@_suffix_tl { - large }
7912     \@@_create_nodes:
7913   \endpgfpicture
7914 }

7915 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7916 {
7917   \pgfpicture
7918     \pgfrememberpicturepositiononpagetrue
7919     \pgf@relevantforpicturesizefalse
7920     \@@_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”.

```

7921   \tl_set:Nn \l_@@_suffix_tl { - medium }
7922   \@@_create_nodes:
7923   \@@_computations_for_large_nodes:
7924   \tl_set:Nn \l_@@_suffix_tl { - large }
7925   \@@_create_nodes:
7926 \endpgfpicture
7927 }

```

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.

```

7928 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7929 {
7930   \int_set:Nn \l_@@_first_row_int 1
7931   \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`.

```

7932   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7933   {
7934     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7935     {
7936       (
7937         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7938         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7939       )
7940       / 2
7941     }
7942     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7943     { l_@@_row _ \@@_i: _ min _ dim }
7944   }
7945   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7946   {
7947     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7948     {
7949       (
7950         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7951         \dim_use:c
7952         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7953       )

```

```

7954         / 2
7955     }
7956     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7957     { l_@@_column _ \@@_j: _ max _ dim }
7958 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7959 \dim_sub:cn
7960 { l_@@_column _ 1 _ min _ dim }
7961 \l_@@_left_margin_dim
7962 \dim_add:cn
7963 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7964 \l_@@_right_margin_dim
7965 }

```

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).

```

7966 \cs_new_protected:Npn \@@_create_nodes:
7967 {
7968     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7969     {
7970         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7971         {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7972             \@@_pgf_rect_node:nnnnn
7973             { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7974             { \dim_use:c { l_@@_column _ \@@_j: _min_dim } }
7975             { \dim_use:c { l_@@_row _ \@@_i: _min_dim } }
7976             { \dim_use:c { l_@@_column _ \@@_j: _max_dim } }
7977             { \dim_use:c { l_@@_row _ \@@_i: _max_dim } }
7978             \str_if_empty:NF \l_@@_name_str
7979             {
7980                 \pgfnodealias
7981                 { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7982                 { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7983             }
7984         }
7985     }
7986     \int_step_inline:nn \c@iRow
7987     {
7988         \pgfnodealias
7989         { \@@_env: - ##1 - last \l_@@_suffix_tl }
7990         { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7991     }
7992     \int_step_inline:nn \c@jCol
7993     {
7994         \pgfnodealias
7995         { \@@_env: - last - ##1 \l_@@_suffix_tl }
7996         { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7997     }
7998     \pgfnodealias
7999     { \@@_env: - last - last \l_@@_suffix_tl }
8000     { \@@_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 .

```

8001     \seq_map_pairwise_function:NNN

```

```

8002 \g_@@_multicolumn_cells_seq
8003 \g_@@_multicolumn_sizes_seq
8004 \@@_node_for_multicolumn:nn
8005 }

8006 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
8007 {
8008   \cs_set_nopar:Npn \@@_i: { #1 }
8009   \cs_set_nopar:Npn \@@_j: { #2 }
8010 }

```

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”).

```

8011 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
8012 {
8013   \@@_extract_coords_values: #1 \q_stop
8014   \@@_pgf_rect_node:nnnnn
8015   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
8016   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
8017   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
8018   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
8019   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
8020   \str_if_empty:NF \l_@@_name_str
8021   {
8022     \pgfnodealias
8023     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
8024     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
8025   }
8026 }

```

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).

```

8027 \keys_define:nn { nicematrix / BlockFirstPass }
8028 {
8029   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
8030   \bool_set_true:N \l_@@_p_block_bool ,
8031   j .value_forbidden:n = true ,
8032   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
8033   l .value_forbidden:n = true ,
8034   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
8035   r .value_forbidden:n = true ,
8036   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
8037   c .value_forbidden:n = true ,
8038   L .code:n = \str_set:Nn \l_@@_hpos_block_str L ,
8039   L .value_forbidden:n = true ,
8040   R .code:n = \str_set:Nn \l_@@_hpos_block_str R ,
8041   R .value_forbidden:n = true ,
8042   C .code:n = \str_set:Nn \l_@@_hpos_block_str C ,
8043   C .value_forbidden:n = true ,
8044   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
8045   t .value_forbidden:n = true ,

```

```

8046 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
8047 T .value_forbidden:n = true ,
8048 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
8049 b .value_forbidden:n = true ,
8050 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
8051 B .value_forbidden:n = true ,
8052 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
8053 m .value_forbidden:n = true ,
8054 v-center .meta:n = m ,
8055 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
8056 p .value_forbidden:n = true ,
8057 respect-arraystretch .code:n =
8058   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
8059 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.

```

8060 color .code:n =
8061   \@@_color:n { #1 }
8062   \tl_set_rescan:Nnn
8063     \l_@@_draw_tl
8064     { \char_set_catcode_other:N ! }
8065     { #1 } ,
8066 color .value_required:n = true ,
8067 }

```

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.

```

8068 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

8069 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
8070 {

```

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).

```

8071   \tl_if_blank:nTF { #2 }
8072     { \@@_Block_ii:nnnnn 1 1 }
8073     {
8074       \tl_if_in:nnTF { #2 } { - }
8075       {
8076         \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
8077         \@@_Block_i_czech:w \@@_Block_i:w
8078         #2 \q_stop
8079       }
8080       {
8081         \@@_error:nn { Bad~argument~for~Block } { #2 }
8082         \@@_Block_ii:nnnnn 1 1
8083       }
8084     }
8085   { #1 } { #3 } { #4 }
8086   \ignorespaces
8087 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

8088 \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`.

```

8089 {
8090   \char_set_catcode_active:N -

```

```

8091 \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
8092 }

```

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.

```

8093 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
8094 {

```

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).

```

8095 \int_set:Nn \l_tmpa_int
8096 {
8097   \bool_lazy_or:nnTF
8098     { \tl_if_blank_p:n { #1 } }
8099     { \str_if_eq_p:ee { * } { #1 } }
8100     { 100 }
8101     { #1 }
8102 }
8103 \int_set:Nn \l_tmpb_int
8104 {
8105   \bool_lazy_or:nnTF
8106     { \tl_if_blank_p:n { #2 } }
8107     { \str_if_eq_p:ee { * } { #2 } }
8108     { 100 }
8109     { #2 }
8110 }

```

If the block is mono-column.

```

8111 \int_compare:nNnTF \l_tmpb_int = 1
8112 {
8113   \tl_if_empty:NTF \l_@@_hpos_cell_tl
8114     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
8115     { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
8116 }
8117 { \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.

```

8118 \keys_set:known:nn { nicematrix / BlockFirstPass } { #3 }
8119 \tl_set:Nn \l_tmpa_tl
8120 {
8121   { \int_use:N \c@iRow }
8122   { \int_use:N \c@jCol }
8123   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
8124   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
8125 }

```

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).

```

8126 \bool_set_false:N \l_tmpa_bool
8127 \bool_if:NT \l_@@_amp_in_blocks_bool

```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

8128     { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
8129     \bool_case:nF
8130     {
8131         \l_tmpa_bool                                { \@@_Block_vii:eennn }
8132         \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.

```

8133         \l_@@_X_bool                                { \@@_Block_v:eennn }
8134         { \tl_if_empty_p:n { #5 } }                  { \@@_Block_v:eennn }
8135         { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
8136         { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
8137     }
8138     { \@@_Block_v:eennn }
8139     { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
8140 }

```

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.

```

8141 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
8142 {
8143     \int_gincr:N \g_@@_block_box_int
8144     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8145     \cs_set_eq:NN \rowcolor \@@_rowcolor_error
8146     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
8147     {
8148         \tl_gput_right:Ne \g_@@_rules_tl
8149         {
8150             \@@_draw_diagbox:nnnnnn
8151             { \int_use:N \c@iRow }
8152             { \int_use:N \c@jCol }
8153             { \int_eval:n { \c@iRow + #1 - 1 } }
8154             { \int_eval:n { \c@jCol + #2 - 1 } }
8155             { \g_@@_row_style_tl \exp_not:n { ##1 } }
8156             { \g_@@_row_style_tl \exp_not:n { ##2 } }
8157         }
8158     }
8159     \box_gclear_new:c
8160     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

8161     \hbox_gset:cn
8162     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
8163     {

```

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`).

```

8164      \tl_if_empty:NTF \l_@@_color_tl
8165      { \int_compare:nNnT { #2 } = 1 \set@color }
8166      { \@@_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`.

```

8167      \int_compare:nNnT { #1 } = 1
8168      {
8169          \int_if_zero:nTF \c@iRow
8170          {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\\
-2 & 3 & -4 & 5 & \\\
3 & -4 & 5 & -6 & \\\
-4 & 5 & -6 & 7 & \\\
5 & -6 & 7 & -8 & \\\
\end{bNiceMatrix}$

```

```

8171      \cs_set_eq:NN \Block \@@_NullBlock:
8172      \l_@@_code_for_first_row_tl
8173    }
8174    {
8175      \int_compare:nNnT \c@iRow = \l_@@_last_row_int
8176      {
8177          \cs_set_eq:NN \Block \@@_NullBlock:
8178          \l_@@_code_for_last_row_tl
8179        }
8180      }
8181      \g_@@_row_style_tl
8182    }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

8183      \@@_reset_arraystretch:
8184      \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

8185      #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.).

```

8186      \@@_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}`.

```

8187 \bool_if:NTF \l_@@_tabular_bool
8188 {
8189   \bool_lazy_all:nTF
8190   {
8191     { \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.

```

8192     { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
8193     { ! \g_@@_rotate_bool }
8194   }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

8195   {
8196     \use:e
8197     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

8198       \exp_not:N \begin { minipage }
8199       [ \str_lowercase:f \l_@@_vpos_block_str ]
8200       { \l_@@_col_width_dim }
8201       \str_case:on \l_@@_hpos_block_str
8202       { c \centering r \raggedleft l \raggedright }
8203     }
8204     #5
8205   \end { minipage }
8206 }

```

In the other cases, we use a `{tabular}`.

```

8207   {
8208     \use:e
8209     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

8210       \exp_not:N \begin { tabular }
8211       [ \str_lowercase:f \l_@@_vpos_block_str ]
8212       { @ { } \l_@@_hpos_block_str @ { } }
8213     }
8214     #5
8215   \end { tabular }
8216 }
8217 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

8218   {
8219     $ % $
8220     \bool_if:NT \l_@@_small_bool
8221     {
8222       \def \arraystretch { 0.47 }
8223       \dim_set:Nn \arraycolsep { 1.45 pt }
8224     }
8225     \use:e
8226     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

8227       \exp_not:N \begin { array }
8228       [ \str_lowercase:f \l_@@_vpos_block_str ]
8229       { @ { } \l_@@_hpos_block_str @ { } }
8230     }
8231     \@@_tuning_key_small:
8232     #5
8233   \end { array }
8234   $ % $
8235 }
8236 }

```

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).

```
8237 \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.

```
8238 \int_compare:nNnT { #2 } = 1
8239 {
8240   \dim_gset:Nn \g_@@_blocks_wd_dim
8241   {
8242     \dim_max:nn
8243     \g_@@_blocks_wd_dim
8244     {
8245       \box_wd:c
8246       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8247     }
8248   }
8249 }
```

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.

```
8250 \int_compare:nNnT { #1 } = 1
8251 {
8252   \bool_lazy_any:nT
8253   {
8254     { \str_if_empty_p:N \l_@@_vpos_block_str }
8255     { \str_if_eq_p:ee t \l_@@_vpos_block_str }
8256     { \str_if_eq_p:ee b \l_@@_vpos_block_str }
8257   }
8258   \@@_adjust_blocks_ht_dp:
8259 }
8260 \seq_gput_right:Ne \g_@@_blocks_seq
8261 {
8262   \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.

```
8263 {
8264   \exp_not:n { #3 } ,
8265   \l_@@_hpos_block_str ,
```

Now, we put a key for the vertical alignment.

```
8266 \bool_if:NT \g_@@_rotate_bool
8267 {
8268   \bool_if:NTF \g_@@_rotate_c_bool
8269   { m }
8270   {
8271     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
8272     { T }
8273   }
8274 }
8275 }
8276 {
8277   \box_use_drop:c
8278   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8279 }
8280 }
8281 \bool_set_false:N \g_@@_rotate_c_bool
8282 }
```

```

8283 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
8284 {
8285   \dim_gset:Nn \g_@@_blocks_ht_dim
8286   {
8287     \dim_max:nn
8288     \g_@@_blocks_ht_dim
8289     {
8290       \box_ht:c
8291       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8292     }
8293   }
8294   \dim_gset:Nn \g_@@_blocks_dp_dim
8295   {
8296     \dim_max:nn
8297     \g_@@_blocks_dp_dim
8298     {
8299       \box_dp:c
8300       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8301     }
8302   }
8303 }

8304 \cs_new:Npn \@@_adjust_hpos_rotate:
8305 {
8306   \bool_if:NT \g_@@_rotate_bool
8307   {
8308     \str_set:Ne \l_@@_hpos_block_str
8309     {
8310       \bool_if:NTF \g_@@_rotate_c_bool
8311       c
8312       {
8313         \str_case:onF \l_@@_vpos_block_str
8314         { b l B l t r T r }
8315         {
8316           \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
8317           r
8318           l
8319         }
8320       }
8321     }
8322   }
8323 }
8324 \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*.

```

8325 \cs_new_protected:Npn \@@_rotate_box_of_block:
8326 {
8327   \box_grotate:cn
8328   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8329   { \bool_if:NTF \g_@@_rotate_minus_bool { -90 } { 90 } }
8330   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
8331   {
8332     \vbox_gset_top:cn
8333     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8334     {
8335       \skip_vertical:n { 0.8 ex }
8336       \box_use:c
8337       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8338     }
8339   }
8340   \bool_if:NT \g_@@_rotate_c_bool

```

```

8341 {
8342   \hbox_gset:cn
8343   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8344   {
8345     \vbox_center:n
8346     {
8347       \box_use:c
8348       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8349     }
8350   }
8351 }
8352 }

```

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.

```

8353 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
8354 {
8355   \seq_gput_right:Ne \g_@@_blocks_seq
8356   {
8357     \l_tmpa_tl
8358     { \exp_not:n { #3 } }
8359     {
8360       \bool_if:NTF \l_@@_tabular_bool
8361       {
8362

```

The following command will be no-op when `respect-arraystretch` is in force.

```

8363 \@@_reset_arraystretch:
8364 \exp_not:n
8365 {
8366   \dim_zero:N \extrarowheight
8367   #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).

```

8368   \tag_if_active:T { \tag_stop:n { table } }
8369   \use:e
8370   {
8371     \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
8372     { @ { } \l_@@_hpos_block_str @ { } }
8373   }
8374   #5
8375   \end { tabular }
8376 }
8377 }
8378 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

8379 {
8380 {

```

The following will be no-op when `respect-arraystretch` is in force.

```

8381 \@@_reset_arraystretch:
8382 \exp_not:n
8383 {
8384   \dim_zero:N \extrarowheight

```

```

8385         #4
8386         $ % $
8387         \use:e
8388         {
8389             \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
8390             { @ { } \l_@@_hpos_block_str @ { } }
8391         }
8392         \@@_tuning_key_small:
8393         #5
8394         \end { array }
8395         $ % $
8396     }
8397 }
8398 }
8399 }
8400 }
8401 }
8402 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

8403 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
8404 {
8405     \seq_gput_right:Ne \g_@@_blocks_seq
8406     {
8407         \l_tmpa_tl
8408         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

8409         { { \exp_not:n { #4 #5 } } }
8410     }
8411 }
8412 \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`.

```

8413 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
8414 {
8415     \seq_gput_right:Ne \g_@@_blocks_seq
8416     {
8417         \l_tmpa_tl
8418         { \exp_not:n { #3 } }
8419         { \exp_not:n { #4 #5 } }
8420     }
8421 }
8422 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }

```

We recall that the options of the command `\Block` are analyzed twice: first in the cell of the array and once again when the block will be put in the array *after the construction of the array* (by using PGF).

```

8423 \keys_define:nn { nicematrix / Block }
8424 {
8425     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
8426     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

8427     tikz .code:n =
8428         \IfPackageLoadedTF { tikz }
8429         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
8430         { \@@_error:n { tikz~key~without~tikz } } ,
8431     tikz .value_required:n = true ,
8432     fill .code:n =
8433         \tl_set_rescan:Nnn

```

```

8434     \l_@@_fill_tl
8435     { \char_set_catcode_other:N ! }
8436     { #1 } ,
8437     fill .value_required:n = true ,

```

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

```

8438     opacity .tl_set:N = \l_@@_opacity_tl ,
8439     opacity .value_required:n = true ,
8440     draw .code:n =
8441     \tl_set_rescan:Nnn
8442     \l_@@_draw_tl
8443     { \char_set_catcode_other:N ! }
8444     { #1 } ,
8445     draw .default:n = default ,
8446     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8447     rounded-corners .default:n = 4 pt ,
8448     color .code:n =
8449     \@@_color:n { #1 }
8450     \tl_set_rescan:Nnn
8451     \l_@@_draw_tl
8452     { \char_set_catcode_other:N ! }
8453     { #1 } ,
8454     borders .clist_set:N = \l_@@_borders_clist ,
8455     borders .default:n = all ,
8456     hvlines .meta:n = { vlines , hlines } ,
8457     vlines .bool_set:N = \l_@@_vlines_block_bool ,
8458     hlines .bool_set:N = \l_@@_hlines_block_bool ,
8459     rules/width .dim_set:N = \arrayrulewidth ,
8460     rules/color .tl_set:N = \l_@@_rules_color_tl ,
8461     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,

```

The key `line-width` is now deprecated (replaced by `rules/width`).

```

8462     line-width .dim_set:N = \arrayrulewidth , % deprecated

```

Some keys have not a property `.value_required:n` (or similar) because they are in `BlockFirstPass`.

```

8463     j .code:n = \str_set:Nn \l_@@_hpos_block_str j
8464     \bool_set_true:N \l_@@_p_block_bool ,
8465     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
8466     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
8467     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
8468     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
8469     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8470     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
8471     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8472     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
8473     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8474     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
8475     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
8476     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
8477     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
8478     m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
8479     m .value_forbidden:n = true ,
8480     v-center .meta:n = m ,
8481     p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
8482     p .value_forbidden:n = true ,
8483     name .str_set:N = \l_@@_block_name_str ,
8484     name .value_required:n = true ,
8485     respect-arraystretch .code:n =
8486     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
8487     respect-arraystretch .value_forbidden:n = true ,
8488     transparent .bool_set:N = \l_@@_transparent_bool ,
8489     unknown .code:n =
8490     \@@_unknown_key:nn
8491     { nicematrix / Block }
8492     { Unknown~key~for~Block }

```

```
8493 }
```

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.

```
8494 \cs_new_protected:Npn \@@_draw_blocks:
8495 {
8496   \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
8497   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
8498 }
8499 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
8500 {
```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```
8501   \int_zero:N \l_@@_last_row_int
8502   \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”).

```
8503   \int_compare:nNnTF { #3 } > { 98 }
8504   { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
8505   { \int_set:Nn \l_@@_last_row_int { #3 } }
8506   \int_compare:nNnTF { #4 } > { 98 }
8507   { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
8508   { \int_set:Nn \l_@@_last_col_int { #4 } }
8509   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
8510   {
8511     \bool_lazy_and:nnTF
8512       \l_@@_preamble_bool
8513       {
8514         \int_compare_p:n
8515           { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
8516       }
8517     { \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 } }
8518     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8519   }
8520   {
8521     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
8522     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8523     {
```

We expand the four first arguments of `\@@_Block_v:nnnnnn`.

```
8524       \use:e
8525       {
8526         \@@_Block_v:nnnnnn
8527         { #1 }
8528         { #2 }
8529         { \int_use:N \l_@@_last_row_int }
8530         { \int_use:N \l_@@_last_col_int }
8531       }
8532       { #5 }
8533       { #6 }
8534     }
8535   }
8536 }
```

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.

```
8537 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
8538 {
```

The group is for the keys.

```
8539 \group_begin:
8540 \int_compare:nNtT { #1 } = { #3 }
8541 { \str_set:Nn \l_@@_vpos_block_str { t } }
8542 \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 than `\str_if_in:nnT`.

```
8543 \bool_if:NT \l_@@_amp_in_blocks_bool
8544 { \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool } }

8545 \bool_lazy_and:nnT
8546 \l_@@_vlines_block_bool
8547 { ! \l_@@_ampersand_bool }
8548 {
8549 \tl_gput_right:Ne \g_@@_rules_tl
8550 {
8551 \@@_vlines_block:nnnnnn
8552 { \dim_use:N \arrayrulewidth }
8553 { \l_@@_rules_color_tl }
8554 { #1 } { #2 } { #3 } { #4 }
8555 }
8556 }

8557 \bool_if:NT \l_@@_hlines_block_bool
8558 {
8559 \tl_gput_right:Ne \g_@@_rules_tl
8560 {
8561 \@@_hlines_block:nnnnnn
8562 { \dim_use:N \arrayrulewidth }
8563 { \l_@@_rules_color_tl }
8564 { #1 } { #2 } { #3 } { #4 }
8565 }
8566 }
```

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).

```
8567 \bool_if:NF \l_@@_transparent_bool
8568 {
8569 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8570 { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
8571 }

8572 \tl_if_empty:NF \l_@@_draw_tl
8573 {
8574 \tl_gput_right:Nn \g_@@_rules_tl
8575 {
8576 \@@_stroke_block:nnnnn
```

#5 are the options

```
8577 { #5 } { #1 } { #2 } { #3 } { #4 }
8578 }
8579 \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
8580 { { #1 } { #2 } { #3 } { #4 } }
8581 }

8582 \clist_if_empty:NF \l_@@_borders_clist
8583 {
8584 \tl_gput_right:Nn \g_nicematrix_code_after_tl
8585 {
```

```

8586         \@@_stroke_borders_block:nnnnn
8587         { #5 } { #1 } { #2 } { #3 } { #4 }
8588     }
8589 }
8590 \tl_if_empty:NF \l_@@_fill_tl
8591 {
8592     \@@_add_opacity_to_fill:
8593     \tl_gput_right:Ne \g_@@_pre_code_before_tl
8594     {
8595         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
8596         { #1 - #2 }
8597         { #3 - #4 }
8598         { \dim_use:N \l_@@_rounded_corners_dim }
8599     }
8600 }
8601 \seq_if_empty:NF \l_@@_tikz_seq
8602 {
8603     \tl_gput_right:Ne \g_nicematrix_code_before_tl
8604     {
8605         \@@_block_tikz:nnnnn
8606         { \seq_use:Nn \l_@@_tikz_seq { , } }
8607         { #1 } { #2 } { #3 } { #4 }

```

We will have in that last field a list of lists of TikZ keys.

```

8608     }
8609 }
8610 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
8611 {
8612     \tl_gput_right:Ne \g_@@_rules_tl
8613     {
8614         \@@_draw_diagbox:nnnnnn
8615         { #1 } { #2 } { #3 } { #4 }
8616         { \exp_not:n { ##1 } }
8617         { \exp_not:n { ##2 } }
8618     }
8619 }

```

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	four	two
six	seven	five
		eight

The construction of the node corresponding to the merged cells.

```

8620 \pgfpicture
8621 \pgfrememberpicturepositiononpagetrue
8622 \pgf@relevantforpicturesizefalse

```

```

8623 \@@_qpoint:n { row - #1 }
8624 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8625 \@@_qpoint:n { col - #2 }
8626 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8627 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
8628 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8629 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8630 \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.

```

8631 \@@_pgf_rect_node:nnnnn
8632 { \@@_env: - #1 - #2 - block }
8633 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8634 \str_if_empty:NF \l_@@_block_name_str
8635 {
8636   \pgfnodealias
8637   { \@@_env: - \l_@@_block_name_str }
8638   { \@@_env: - #1 - #2 - block }
8639   \str_if_empty:NF \l_@@_name_str
8640   {
8641     \pgfnodealias
8642     { \l_@@_name_str - \l_@@_block_name_str }
8643     { \@@_env: - #1 - #2 - block }
8644   }
8645 }

```

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.

```

8646 \bool_if:NF \l_@@_hpos_of_block_cap_bool
8647 {
8648   \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.

```

8649 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8650 {

```

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.

```

8651 \cs_if_exist:cT
8652 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8653 {
8654   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8655   {
8656     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
8657     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
8658   }
8659 }
8660 }

```

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.

```

8661 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
8662 {
8663   \@@_qpoint:n { col - #2 }
8664   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8665 }
8666 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
8667 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8668 {

```

```

8669 \cs_if_exist:cT
8670 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8671 {
8672 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8673 {
8674 \pgfpointanchor
8675 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8676 { east }
8677 \dim_set:Nn \l_@@_tmpd_dim
8678 { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
8679 }
8680 }
8681 }
8682 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
8683 {
8684 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8685 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8686 }
8687 \@@_pgf_rect_node:nnnnn
8688 { \@@_env: - #1 - #2 - block - short }
8689 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8690 }

```

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.

```

8691 \bool_if:NT \l_@@_medium_nodes_bool
8692 {
8693 \@@_pgf_rect_node:nnn
8694 { \@@_env: - #1 - #2 - block - medium }
8695 { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
8696 {
8697 \pgfpointanchor
8698 { \@@_env:
8699 - \int_use:N \l_@@_last_row_int
8700 - \int_use:N \l_@@_last_col_int - medium
8701 }
8702 { south-east }
8703 }
8704 }
8705 \endpgfpicture
8706

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand &.

```

8707 \bool_if:NtF \l_@@_ampersand_bool
8708 {
8709 \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
8710 \int_zero_new:N \l_@@_split_int
8711 \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).

```

8712 \int_set:Nn \l_@@_first_row_int { #1 }
8713 \int_set:Nn \l_@@_first_col_int { #2 }
8714 \int_set:Nn \l_@@_last_row_int { #3 }
8715 \int_set:Nn \l_@@_last_col_int { #4 }
8716 \pgfpicture
8717 \pgfrememberpicturepositiononpagetrue
8718 \pgf@relevantforpicturesizefalse
8719 \@@_qpoint:n { row - #1 }
8720 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8721 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8722 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y

```

```

8723 \@@_qpoint:n { col - #2 }
8724 \dim_set_eq:NN \l_tmpa_dim \pgf@x
8725 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8726 \dim_set:Nn \l_tmpb_dim
8727 { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
8728 \bool_lazy_or:nnT
8729 \l_@@_vlines_block_bool
8730 { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
8731 {
8732   \int_step_inline:nn { \l_@@_split_int - 1 }
8733   {
8734     \pgfpathmoveto
8735     {
8736       \pgfpoint
8737       { \l_tmpa_dim + ##1 \l_tmpb_dim }
8738       \l_@@_tmpc_dim
8739     }
8740     \pgfpathlineto
8741     {
8742       \pgfpoint
8743       { \l_tmpa_dim + ##1 \l_tmpb_dim }
8744       \l_@@_tmpd_dim
8745     }
8746     \CT@arc@
8747     \pgfsetlinewidth { 1.1 \arrayrulewidth }
8748     \pgfsetrectcap
8749     \pgfusepathqstroke
8750   }
8751 }
8752 \cs_set_eq:NN \cellcolor \@@_subcellcolor
8753 \int_zero_new:N \l_@@_split_i_int
8754 \str_if_eq:eeTF \l_@@_vpos_block_str T
8755 {
8756   \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
8757   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8758 }
8759 {
8760   \str_if_eq:eeTF \l_@@_vpos_block_str B
8761   {
8762     \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
8763     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8764   }
8765   {
8766     \bool_lazy_or:nnTF
8767     { \int_compare_p:nNn { #1 } = { #3 } }
8768     { \str_if_eq_p:ee \l_@@_vpos_block_str t }
8769     {
8770       \@@_qpoint:n { row - #1 - base }
8771       \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8772     }
8773     {
8774       \str_if_eq:eeTF \l_@@_vpos_block_str b
8775       {
8776         \@@_qpoint:n { row - #3 - base }
8777         \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8778       }
8779       {
8780         \@@_qpoint:n { #1 - #2 - block }
8781         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8782       }
8783     }
8784   }
8785 }

```

```

8786 \int_step_inline:nn \l_@@_split_int
8787 {
8788   \group_begin:

```

The counter `\l_@@_split_i_int` is only for the command `\@@_subcellcolor`.

```

8789   \int_set:Nn \l_@@_split_i_int { ##1 }
8790   \dim_set:Nn \col@sep
8791     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
8792   \pgftransformshift
8793     {
8794     \pgfpoint
8795     {
8796       \l_tmpa_dim + ##1 \l_tmpb_dim -
8797       \str_case:on \l_@@_hpos_block_str
8798         {
8799           l { \l_tmpb_dim + \col@sep }
8800           c { 0.5 \l_tmpb_dim }
8801           r { \col@sep }
8802         }
8803       }
8804     { \l_@@_tmpc_dim }
8805   }
8806   \pgfset { inner~sep = \c_zero_dim }
8807   \pgfnode
8808     { rectangle }
8809     {
8810       \str_if_eq:eeTF T \l_@@_vpos_block_str
8811       {
8812         \str_case:on \l_@@_hpos_block_str
8813         {
8814           l { north-west }
8815           c { north }
8816           r { north-east }
8817         }
8818       }
8819     {
8820       \str_if_eq:eeTF B \l_@@_vpos_block_str
8821       {
8822         \str_case:on \l_@@_hpos_block_str
8823         {
8824           l { south-west }
8825           c { south }
8826           r { south-east }
8827         }
8828       }
8829     {
8830       \bool_lazy_any:nTF
8831       {
8832         { \int_compare_p:nNn { #1 } = { #3 } }
8833         { \str_if_eq_p:ee t \l_@@_vpos_block_str }
8834         { \str_if_eq_p:ee b \l_@@_vpos_block_str }
8835       }
8836       {
8837         \str_case:on \l_@@_hpos_block_str
8838         {
8839           l { base-west }
8840           c { base }
8841           r { base-east }
8842         }
8843       }
8844     {
8845       \str_case:on \l_@@_hpos_block_str
8846       {
8847         l { west }

```

```

8848             c { center }
8849             r { east }
8850         }
8851     }
8852 }
8853 }
8854 }
8855 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8856 \group_end:
8857 }
8858 \endpgfpicture
8859 }

```

Now the case where there is no ampersand & in the content of the block.

```

8860 {
8861   \bool_if:NTF \l_@@_p_block_bool
8862   {

```

When the final user has used the key p, we have to compute the width.

```

8863     \pgfpicture
8864     \pgfrememberpicturepositiononpagetrue
8865     \pgf@relevantforpicturesizefalse
8866     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8867     {
8868       \@@_qpoint:n { col - #2 }
8869       \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8870       \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8871     }
8872     {
8873       \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8874       \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8875       \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8876     }
8877     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8878     \endpgfpicture
8879     \hbox_set:Nn \l_@@_cell_box
8880     {
8881       \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8882         { \g_tmpb_dim }
8883       \str_case:on \l_@@_hpos_block_str
8884         { c \centering r \raggedleft l \raggedright j { } }
8885       \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8886       #6
8887       \end { minipage }
8888     }
8889   }
8890   {
8891     \hbox_set:Nn \l_@@_cell_box
8892     {
8893       \set@color
8894       \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8895       #6
8896     }
8897   }
8898   \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.

```

8899     \pgfpicture
8900     \pgfrememberpicturepositiononpagetrue
8901     \pgf@relevantforpicturesizefalse
8902     \bool_lazy_any:nTF
8903     {
8904       { \str_if_empty_p:N \l_@@_vpos_block_str }

```

```

8905         { \str_if_eq_p:ee c \l_@@_vpos_block_str }
8906         { \str_if_eq_p:ee T \l_@@_vpos_block_str }
8907         { \str_if_eq_p:ee B \l_@@_vpos_block_str }
8908     }

```

```

8909     {

```

If we are in the “first column”, we must put the block as if it was with the key r.

```

8910         \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.

```

8911         \bool_if:nT \g_@@_last_col_found_bool
8912         {
8913             \int_compare:nNnT { #2 } = \g_@@_col_total_int
8914             { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8915         }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8916         \tl_set:Ne \l_tmpa_tl
8917         {
8918             \str_case:on \l_@@_vpos_block_str
8919             {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8920             { } {
8921                 \str_case:on \l_@@_hpos_block_str
8922                 {
8923                     c { center }
8924                     l { west }
8925                     r { east }
8926                     j { center }
8927                 }
8928             }
8929             c {
8930                 \str_case:on \l_@@_hpos_block_str
8931                 {
8932                     c { center }
8933                     l { west }
8934                     r { east }
8935                     j { center }
8936                 }
8937             }
8938         }
8939         T {
8940             \str_case:on \l_@@_hpos_block_str
8941             {
8942                 c { north }
8943                 l { north~west }
8944                 r { north~east }
8945                 j { north }
8946             }
8947         }
8948     }
8949     B {
8950         \str_case:on \l_@@_hpos_block_str
8951         {
8952             c { south }
8953             l { south~west }
8954             r { south~east }
8955             j { south }
8956         }
8957     }
8958 }
8959 }

```

```

8960     }
8961     \pgftransformshift
8962     {
8963         \pgfpointanchor
8964         {
8965             \@@_env: - #1 - #2 - block
8966             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8967         }
8968         \l_tmpa_tl
8969     }
8970     \pgfset { inner~sep = \c_zero_dim }
8971     \pgfnode
8972     { rectangle }
8973     \l_tmpa_tl
8974     { \box_use_drop:N \l_@@_cell_box } { } { }
8975 }

```

End of the case when `\l_@@_vpos_block_str` is equal to c, T or B. Now, the other cases.

```

8976 {
8977     \pgfextracty \l_tmpa_dim
8978     {
8979         \@@_qpoint:n
8980         {
8981             row - \str_if_eq:eeTF b \l_@@_vpos_block_str { #3 } { #1 }
8982             - base
8983         }
8984     }
8985     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

8986     \pgfpointanchor
8987     {
8988         \@@_env: - #1 - #2 - block
8989         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8990     }
8991     {
8992         \str_case:on \l_@@_hpos_block_str
8993         {
8994             c { center }
8995             l { west }
8996             r { east }
8997             j { center }
8998         }
8999     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

9000     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
9001     \pgfset { inner~sep = \c_zero_dim }
9002     \pgfnode
9003     { rectangle }
9004     {
9005         \str_case:on \l_@@_hpos_block_str
9006         {
9007             c { base }
9008             l { base~west }
9009             r { base~east }
9010             j { base }
9011         }
9012     }
9013     { \box_use_drop:N \l_@@_cell_box } { } { }
9014 }
9015 \endpgfpicture
9016 }

```

```

9017 \group_end:
9018 }
9019 \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).

```

9020 \cs_new_protected:Npn \@@_add_opacity_to_fill:
9021 {
9022   \tl_if_empty:NF \l_@@_opacity_tl
9023   {
9024     \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
9025       {
9026         \tl_set:Nx \l_@@_fill_tl
9027         {
9028           [ opacity = \l_@@_opacity_tl ,
9029             \tl_tail:o \l_@@_fill_tl
9030         ]
9031       }
9032     {
9033       \tl_set:Nx \l_@@_fill_tl
9034       { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
9035     }
9036   }
9037 }

```

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*.

```

9038 \cs_new_protected:Npn \@@_stroke_block:nnnnn #1 #2 #3 #4 #5
9039 {
9040   \group_begin:
9041   \tl_clear:N \l_@@_draw_tl
9042   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
9043   \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
9044   \tl_if_empty:NF \l_@@_draw_tl
9045   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

9046     \tl_if_eq:NnTF \l_@@_draw_tl { default }
9047     \CT@arc@
9048     { \@@_color:o \l_@@_draw_tl }
9049   }

```

The following code can't be put just before the `\pgfusepath { stroke }` (it's too late).

```

9050 \pgfsetcornersarced
9051 { \pgfpoint \l_@@_rounded_corners_dim \l_@@_rounded_corners_dim }
9052 \int_compare:nNnF { #2 } > \c@iRow
9053 {
9054   \int_compare:nNnF { #3 } > \c@jCol
9055   {
9056     \@@_qpoint:n { row - #2 }
9057     \dim_set_eq:NN \l_tmpb_dim \pgf@y
9058     \@@_qpoint:n { col - #3 }
9059     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
9060     \@@_qpoint:n
9061     { row - \int_eval:n { 1 + \int_min:nn \c@iRow { #4 } } }
9062     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9063     \@@_qpoint:n
9064     { col - \int_eval:n { 1 + \int_min:nn \c@jCol { #5 } } }
9065     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
9066     \pgfpathrectanglecorners
9067     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }

```

```

9068         { \pgfpoint \pgf@x \l_tmpa_dim }
9069         \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
9070         \pgfusepathqstroke
9071         { \pgfusepath { stroke } }
9072     }
9073 }
9074 \group_end:
9075 }

```

Here is the set of keys for the command `\@@_stroke_block:nnnnn`.

```

9076 \keys_define:nn { nicematrix / BlockStroke }
9077 {
9078     color .tl_set:N = \l_@@_draw_tl ,
9079     draw .code:n =
9080         \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
9081     draw .default:n = default ,
9082     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
9083     rounded-corners .default:n = 4 pt ,
9084     rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The key `line-width` is now deprecated.

```

9085     line-width .dim_set:N = \l_@@_line_width_dim % deprecated
9086 }

```

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 *th* color. The other arguments are the position of the block: *imin*, *jmin*, *imax* and *jmax*.

```

9087 \cs_new_protected:Npn \@@_vlines_block:nnnnnn #1 #2 #3 #4 #5 #6
9088 {
9089     \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).

```

9090     \dim_set:Nn \arrayrulewidth { #1 }
9091     \pgfsetlinewidth \arrayrulewidth
9092     \@@_set_CTarc:n { #2 }
9093     \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).

```

9094     \seq_set_filter:NnN \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
9095     {
9096         \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #3 }
9097         ||
9098         \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #4 }
9099         ||
9100         \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #5 }
9101         ||
9102         \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #6 }
9103     }
9104     \bool_if:NT \l_@@_fix_vertex_bool
9105     {
9106         \int_compare:nNnT { #4 } > 1
9107         {
9108             \seq_put_right:Ne \g_@@_pos_of_blocks_seq
9109             {
9110                 { 1 }
9111                 { 1 }
9112                 { \int_use:N \c@iRow }
9113                 { \int_eval:n { #4 -1 } }

```

```

9114         { }
9115     }
9116 }
9117 \int_compare:nNt { #6 } < \c@jCol
9118 {
9119     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
9120     {
9121         { 1 }
9122         { \int_eval:n { #6 + 1 } }
9123         { \int_use:N \c@iRow }
9124         { \int_use:N \c@jCol }
9125     }
9126 }
9127 }
9128 }
9129 \int_set:Nn \l_@@_start_int { #3 }
9130 \int_set:Nn \l_@@_end_int { #5 }
9131 \int_step_inline:nnn { #4 } { #6 + 1 }
9132 {
9133     \int_set:Nn \l_@@_position_int { ##1 }
9134     \socket_use:n { nicematrix / draw-vrule }
9135 }
9136 \group_end:
9137 }

```

The command `\@@_hlines_block:nnnnnn` is used only once: in `\@@_Block_v:nnnnnn` which actually draws the block.

```

9138 \cs_new_protected:Npn \@@_hlines_block:nnnnnn #1 #2 #3 #4 #5 #6
9139 {
9140     \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).

```

9141     \dim_set:Nn \arrayrulewidth { #1 }
9142     \pgfsetlinewidth \arrayrulewidth
9143     \@@_set_CTarc:n { #2 }
9144     \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).

```

9145     \seq_set_filter:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
9146     {
9147         \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #3 }
9148         ||
9149         \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #4 }
9150         ||
9151         \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #5 }
9152         ||
9153         \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #6 }
9154     }
9155     \bool_if:NT \l_@@_fix_vertex_bool
9156     {
9157         \int_compare:nNt { #3 } > 1
9158         {
9159             \seq_put_right:Ne \g_@@_pos_of_blocks_seq
9160             {
9161                 { 1 }
9162                 { 1 }
9163                 { \int_eval:n { #3 - 1 } }
9164                 { \int_use:N \c@jCol }

```

```

9165         { }
9166     }
9167 }
9168 \int_compare:nNtT { #5 } < \c@iRow
9169 {
9170     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
9171     {
9172         { \int_eval:n { #5 + 1 } }
9173         { 1 }
9174         { \int_use:N \c@iRow }
9175         { \int_use:N \c@jCol }
9176         { }
9177     }
9178 }
9179 }
9180 \int_set:Nn \l_@@_start_int { #4 }
9181 \int_set:Nn \l_@@_end_int { #6 }
9182 \int_step_inline:nnn { #3 } { #5 + 1 }
9183 {
9184     \int_set:Nn \l_@@_position_int { ##1 }
9185     \socket_use:n { nicematrix / draw-hrule }
9186 }
9187 \group_end:
9188 }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke.

```

9189 \cs_new_protected:Npn \@@_stroke_borders_block:nnnnn #1 #2 #3 #4 #5
9190 {
9191     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
9192     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
9193
9194     \dim_compare:nNtTF \l_@@_rounded_corners_dim > \c_zero_dim
9195     { \@@_error:n { borders~forbidden } }
9196     {
9197         \tl_clear_new:N \l_@@_borders_tikz_tl
9198         \keys_set:no { nicematrix / OnlyForTikzInBorders } \l_@@_borders_clist
9199         \tl_set:Nn \l_@@_tmpc_tl { #2 }
9200         \tl_set:Nn \l_@@_tmpd_tl { #3 }
9201         \tl_set:Ne \l_tmpa_tl { \int_eval:n { #4 + 1 } }
9202         \tl_set:Ne \l_tmpb_tl { \int_eval:n { #5 + 1 } }
9203         \@@_stroke_borders_block_i:
9204     }
9205 }
9206 \AtBeginDocument
9207 {
9208     \cs_new_protected:Npe \@@_stroke_borders_block_i:
9209     {
9210         \c_@@_pgfortikzpicture_tl
9211         \@@_stroke_borders_block_ii:
9212         \c_@@_endpgfortikzpicture_tl
9213     }
9214 }
9215 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
9216 {
9217     \pgfrememberpicturepositiononpagetrue
9218     \pgf@relevantforpicturesizefalse
9219     \CT@arc@
9220     \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).

```

9221 \bool_set_false:N \l_tmpa_bool
9222 \clist_if_in:NnT \l_@@_borders_clist { right }
9223 {
9224   \@@_stroke_vertical:n \l_tmpb_tl
9225   \bool_set_true:N \l_tmpa_bool
9226 }
9227 \clist_if_in:NnT \l_@@_borders_clist { left }
9228 {
9229   \@@_stroke_vertical:n \l_@@_tmpd_tl
9230   \bool_set_true:N \l_tmpa_bool
9231 }
9232 \clist_if_in:NnT \l_@@_borders_clist { bottom }
9233 {
9234   \@@_stroke_horizontal:n \l_tmpa_tl
9235   \bool_set_true:N \l_tmpa_bool
9236 }
9237 \clist_if_in:NnT \l_@@_borders_clist { top }
9238 {
9239   \@@_stroke_horizontal:n \l_@@_tmpc_tl
9240   \bool_set_true:N \l_tmpa_bool
9241 }
9242 \bool_if:NF \l_tmpa_bool
9243 {
9244   \@@_stroke_vertical:n \l_tmpb_tl
9245   \@@_stroke_vertical:n \l_@@_tmpd_tl
9246   \@@_stroke_horizontal:n \l_tmpa_tl
9247   \@@_stroke_horizontal:n \l_@@_tmpc_tl
9248 }
9249 }
9250 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
9251 {
9252   tikz .code:n =
9253     \cs_if_exist:NTF \tikzpicture
9254     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
9255     { \@@_error:n { tikz~in~borders~without~tikz } } ,
9256   tikz .value_required:n = true ,
9257   dashed .meta:n = { tikz = dashed } ,
9258   top .code:n = ,
9259   bottom .code:n = ,
9260   left .code:n = ,
9261   right .code:n = ,
9262   all .code:n = ,
9263   unknown .code:n = \@@_error:n { bad~border }
9264 }

```

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).

```

9265 \cs_new_protected:Npn \@@_stroke_vertical:n #1
9266 {
9267   \@@_qpoint:n \l_@@_tmpc_tl
9268   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
9269   \@@_qpoint:n \l_tmpa_tl
9270   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
9271   \@@_qpoint:n { #1 }
9272   \tl_if_empty:NTF \l_@@_borders_tikz_tl
9273   {
9274     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
9275     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
9276     \pgfusepathqstroke
9277   }
9278   {
9279     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
9280     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;

```

```

9281     }
9282 }

```

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).

```

9283 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
9284 {
9285   \@@_qpoint:n \l_@@_tmpd_tl
9286   \clist_if_in:NnTF \l_@@_borders_clist { left }
9287     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
9288     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
9289   \@@_qpoint:n \l_tmpb_tl
9290   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
9291   \@@_qpoint:n { #1 }
9292   \tl_if_empty:NTF \l_@@_borders_tikz_tl
9293     {
9294       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
9295       \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9296       \pgfusepathqstroke
9297     }
9298     {
9299       \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
9300         ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
9301     }
9302 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

9303 \keys_define:nn { nicematrix / BlockBorders }
9304 {
9305   borders .clist_set:N = \l_@@_borders_clist ,
9306   borders .default:n = all ,
9307   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
9308   rounded-corners .default:n = 4 pt ,
9309   rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The following key is deprecated.

```

9310   line-width .dim_set:N = \l_@@_line_width_dim % deprecated
9311 }

```

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.

```

9312 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
9313 {
9314   \begin { tikzpicture }
9315   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because #5 is a list of lists.

```

9316   \clist_map_inline:nn { #1 }
9317   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

9318     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
9319     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
9320     (
9321     [
9322       xshift = \dim_use:N \l_@@_xoffset_dim ,
9323       yshift = - \dim_use:N \l_@@_yoffset_dim

```

```

9324         ]
9325         #2 -| #3
9326     )
9327     rectangle
9328     (
9329     [
9330         xshift = - \dim_use:N \l_@@_xoffset_dim ,
9331         yshift = \dim_use:N \l_@@_yoffset_dim
9332     ]
9333     \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
9334     ) ;
9335 }
9336 \end { tikzpicture }
9337 }
9338 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

9339 \keys_define:nn { nicematrix / SpecialOffset }
9340 {
9341     xoffset .dim_set:N = \l_@@_xoffset_dim ,
9342     yoffset .dim_set:N = \l_@@_yoffset_dim ,
9343     offset .meta:n = { xoffset = #1 , yoffset = #1 }
9344 }

```

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.

```

9345 \cs_new_protected:Npn \@@_NullBlock:
9346 { \@@_collect_options:n { \@@_NullBlock_i: } }
9347 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
9348 { }

```

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.

```

9349 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
9350 {
9351     \tl_gput_right:Ne \g_@@_pre_code_before_tl
9352     {

```

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`).

```

9353     \@@_subcellcolor:nnnnnnn
9354     {
9355         \tl_if_blank:nTF { #1 }
9356         { { \exp_not:n { #2 } } }
9357         { [ #1 ] { \exp_not:n { #2 } } }
9358     }
9359     { \int_use:N \l_@@_first_row_int } % first row of the block
9360     { \int_use:N \l_@@_first_col_int } % first column of the block
9361     { \int_use:N \l_@@_last_row_int } % last row of the block
9362     { \int_use:N \l_@@_last_col_int } % last column of the block
9363     { \int_use:N \l_@@_split_int }
9364     { \int_use:N \l_@@_split_i_int }
9365 }
9366 \ignorespaces
9367 }

9368 \cs_new_protected:Npn \@@_subcellcolor:nnnnnnn #1 #2 #3 #4 #5 #6 #7
9369 {
9370     \@@_color_opacity: #1
9371     \pgfpicture
9372     \pgf@relevantforpicturesizefalse
9373     \@@_qpoint:n { col - #3 }

```

```

9374 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
9375 @@_qpoint:n { col - \int_eval:n { #5 + 1 } }
9376 \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
9377 \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
9378 @@_qpoint:n { row - \int_eval:n { #4 + 1 } }
9379 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
9380 @@_qpoint:n { row - #2 }
9381 \pgfpathrectanglecorners
9382 { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } \l_@@_tmpc_dim }
9383 { \pgfpoint \l_tmpb_dim \pgf@y }
9384 \pgfusepathqfill
9385 \endpgfpicture
9386 }

```

27 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

9387 \keys_define:nn { nicematrix / Auto }
9388 {
9389   columns-type .tl_set:N = \l_@@_columns_type_tl ,
9390   columns-type .value_required:n = true ,
9391   l .meta:n = { columns-type = l } ,
9392   r .meta:n = { columns-type = r } ,
9393   c .meta:n = { columns-type = c } ,
9394   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9395   delimiters / color .value_required:n = true ,
9396   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
9397   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
9398   delimiters .value_required:n = true ,
9399   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
9400   rounded-corners .default:n = 4 pt
9401 }

9402 \NewDocumentCommand \AutoNiceMatrixWithDelims
9403 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
9404 { @@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

9405 \cs_new_protected:Npn @@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
9406 {

```

The group is for the protection of the keys.

```

9407 \group_begin:
9408 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
9409 \use:e
9410 {
9411   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
9412   { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
9413   [ \exp_not:o \l_tmpa_tl ]
9414 }
9415 \int_if_zero:nT \l_@@_first_row_int
9416 {
9417   \int_if_zero:nT \l_@@_first_col_int { & }
9418   \prg_replicate:nn { #4 - 1 } { & }
9419   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9420 }
9421 \prg_replicate:nn { #3 }
9422 {
9423   \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`).

```

9424     \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
9425     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9426   }
9427   \int_compare:nNnT \l_@@_last_row_int > { -2 }
9428   {
9429     \int_if_zero:nT \l_@@_first_col_int { & }
9430     \prg_replicate:nn { #4 - 1 } { & }
9431     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9432   }
9433   \end { NiceArrayWithDelims }
9434   \group_end:
9435 }

9436 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
9437 {
9438   \cs_set_protected:cpn { #1 AutoNiceMatrix }
9439   {
9440     \bool_gset_true:N \g_@@_delims_bool
9441     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
9442     \AutoNiceMatrixWithDelims { #2 } { #3 }
9443   }
9444 }

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

9445 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
9446 {
9447   \group_begin:
9448   \bool_gset_false:N \g_@@_delims_bool
9449   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
9450   \group_end:
9451 }

```

28 The redefinition of the command `\dotfill`

```

9452 \cs_set_eq:NN \@@_old_dotfill: \dotfill
9453 \cs_new_protected:Npn \@@_dotfill:
9454 {

```

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}`).

```

9455   \@@_old_dotfill:
9456   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
9457 }

```

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`.

```

9458 \cs_new_protected:Npn \@@_dotfill_i:
9459 {
9460   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim
9461     { \@@_old_dotfill: }
9462 }

```

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.

```

9463 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
9464 {
9465   \tl_gput_right:Ne \g_@@_rules_tl
9466   {
9467     \@@_draw_diagbox:nnnnnn
9468     { \int_use:N \c@iRow }
9469     { \int_use:N \c@jCol }
9470     { \int_use:N \c@iRow }
9471     { \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.

```

9472     { \g_@@_row_style_tl \exp_not:n { #1 } }
9473     { \g_@@_row_style_tl \exp_not:n { #2 } }
9474   }

```

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`.

```

9475   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
9476   {
9477     { \int_use:N \c@iRow }
9478     { \int_use:N \c@jCol }
9479     { \int_use:N \c@iRow }
9480     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

9481     { }
9482   }
9483 }

```

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.

```

9484 \cs_new_protected:Npn \@@_draw_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
9485 {

```

We *must* use an environment `{pgfscope}` and not a simple group of TeX.

```

9486   \begin { pgfscope }
9487   \@@_qpoint:n { row - #1 }
9488   \dim_set_eq:NN \l_tmpa_dim \pgf@y
9489   \@@_qpoint:n { col - #2 }
9490   \dim_set_eq:NN \l_tmpb_dim \pgf@x
9491   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
9492   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
9493   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
9494   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
9495   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
9496   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
9497   {

```

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.

```

9498     \CT@arc@
9499     \pgfsetroundcap
9500     \pgfusepathqstroke
9501   }
9502   \pgfset { inner~sep = 1 pt }

```

```

9503 \pgfscope
9504 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
9505 \pgfnode { rectangle } { south~west }
9506 {
9507 \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

9508 \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
9509 \end { minipage }
9510 }
9511 { }
9512 { \pgfusepath { discard } } % mandatory
9513 \endpgfscope
9514 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
9515 \pgfnode { rectangle } { north~east }
9516 {
9517 \begin { minipage } { 20 cm }
9518 \raggedleft
9519 \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
9520 \end { minipage }
9521 }
9522 { }
9523 { \pgfusepath { discard } } % mandatory
9524 \end { pgfscope }
9525 }

```

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. 92.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

9526 \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 `\`.

```

9527 \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`.

```

9528 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
9529 {
9530 \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
9531 \@@_CodeAfter_iv:n
9532 }

```

We catch the argument of the command `\end` (in #1).

```

9533 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
9534 {

```

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.

```

9535 \str_if_eq:eeTF \@currentenv { #1 }
9536 { \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`.

```

9537     {
9538         \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
9539         \@@_CodeAfter_ii:n
9540     }
9541 }

```

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.

```

9542 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
9543 {
9544     \pgfpicture
9545     \pgfrememberpicturepositiononpagetrue
9546     \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.

```

9547     \@@_qpoint:n { row - 1 }
9548     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - \arrayrulewidth }
9549     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
9550     \dim_set:Nn \l_@@_y_final_dim { \pgf@y - \arrayrulewidth }

```

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).

```

9551     \bool_if:nTF { #3 }
9552     { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
9553     { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
9554     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
9555     {
9556         \cs_if_exist:cT
9557         { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
9558         {
9559             \pgfpointanchor
9560             { \@@_env: - ##1 - #2 }
9561             { \bool_if:nTF { #3 } { west } { east } }
9562             \dim_set:Nn \l_tmpa_dim
9563             {
9564                 \bool_if:nTF { #3 }
9565                 \dim_min:nn
9566                 \dim_max:nn
9567                 \l_tmpa_dim
9568                 \pgf@x
9569             }
9570         }
9571     }

```

Now we can put the delimiter with a node of PGF.

```

9572 \pgfset { inner~sep = \c_zero_dim }
9573 \dim_zero:N \nulldelimiterspace
9574 \pgftransformshift
9575 {
9576   \pgfpoint
9577   \l_tmpa_dim
9578   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
9579 }
9580 \pgfnode
9581 { rectangle }
9582 { \bool_if:nTF { #3 } { east } { west } }
9583 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

9584 \nullfont
9585 $ % $
9586 \@@_color:o \l_@@_delimiters_color_tl
9587 \bool_if:nTF { #3 } { \left #1 } { \left . }
9588 \vcenter
9589 {
9590   \nullfont
9591   \hrule \@height
9592   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
9593   \@depth \c_zero_dim
9594   \@width \c_zero_dim
9595 }
9596 \bool_if:nTF { #3 } { \right . } { \right #1 }
9597 $ % $
9598 }
9599 { }
9600 { }
9601 \endpgfpicture
9602 }

```

32 The command \SubMatrix

```

9603 \keys_define:nn { nicematrix / sub-matrix }
9604 {
9605   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
9606   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
9607   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
9608   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
9609   xshift .value_required:n = true ,
9610   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9611   delimiters / color .value_required:n = true ,
9612   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
9613   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9614   hlines .default:n = all ,
9615   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9616   vlines .default:n = all ,
9617   hvlines .meta:n = { hlines, vlines } ,
9618   hvlines .value_forbidden:n = true
9619 }
9620 \keys_define:nn { nicematrix }
9621 {
9622   SubMatrix .inherit:n = nicematrix / sub-matrix ,
9623   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9624   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9625   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9626 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

9627 \keys_define:nn { nicematrix / SubMatrix }
9628 {
9629   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9630   delimiters / color .value_required:n = true ,
9631   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9632   hlines .default:n = all ,
9633   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9634   vlines .default:n = all ,
9635   hvlines .meta:n = { hlines, vlines } ,
9636   hvlines .value_forbidden:n = true ,
9637   name .code:n =
9638     \tl_if_empty:nTF { #1 }
9639     { \@@_error:n { Invalid-name } }
9640     {
9641       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
9642       {
9643         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
9644         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
9645         {
9646           \str_set:Nn \l_@@_submatrix_name_str { #1 }
9647           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
9648         }
9649       }
9650       { \@@_error:n { Invalid-name } }
9651     } ,
9652   name .value_required:n = true ,
9653   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
9654   rules .value_required:n = true ,
9655   code .tl_set:N = \l_@@_code_tl ,
9656   code .value_required:n = true ,
9657   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
9658 }

9659 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
9660 {
9661   \tl_gput_right:Ne \g_@@_pre_code_after_tl
9662   {
9663     \SubMatrix { #1 } { #2 } { #3 } { #4 }
9664     [
9665       delimiters / color = \l_@@_delimiters_color_tl ,
9666       hlines = \l_@@_submatrix_hlines_clist ,
9667       vlines = \l_@@_submatrix_vlines_clist ,
9668       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
9669       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
9670       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
9671       slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
9672       #5
9673     ]
9674   }
9675   \@@_SubMatrix_in_code_before_i { #2 } { #3 }
9676   \ignorespaces
9677 }

9678 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
9679 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9680 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

9681 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
9682 {
9683   \seq_gput_right:Ne \g_@@_submatrix_seq
9684   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

9685 { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
9686 { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
9687 { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
9688 { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
9689 }
9690 }

```

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).

```

9691 \NewDocumentCommand \@@_compute_i_j:nn
9692 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9693 { \@@_compute_i_j:nnnn #1 #2 }

9694 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
9695 {
9696   \def \l_@@_first_i_tl { #1 }
9697   \def \l_@@_first_j_tl { #2 }
9698   \def \l_@@_last_i_tl { #3 }
9699   \def \l_@@_last_j_tl { #4 }
9700   \tl_if_eq:NnT \l_@@_first_i_tl { last }
9701     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
9702   \tl_if_eq:NnT \l_@@_first_j_tl { last }
9703     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
9704   \tl_if_eq:NnT \l_@@_last_i_tl { last }
9705     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
9706   \tl_if_eq:NnT \l_@@_last_j_tl { last }
9707     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
9708 }

```

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`.

```

9709 \AtBeginDocument
9710 {
9711   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m O { } E { _ ^ } { { } { } } }
9712   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
9713     { \@@_sub_matrix:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
9714 }

9715 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
9716 {
9717   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

9718 \@@_compute_i_j:nn { #2 } { #3 }
9719 \int_compare:NnT \l_@@_first_i_tl = \l_@@_last_i_tl
9720 { \def \arraystretch { 1 } }
9721 \bool_lazy_or:nnTF

```

```

9722 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9723 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9724 { \@@_error:nn { Construct~too~large } { \SubMatrix } }
9725 {
9726   \str_clear_new:N \l_@@_submatrix_name_str
9727   \keys_set:nn { nicematrix / SubMatrix } { #5 }
9728   \pgfpicture
9729   \pgfrememberpicturepositiononpagetrue
9730   \pgf@relevantforpicturesizefalse
9731   \pgfset { inner~sep = \c_zero_dim }
9732   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9733   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curryfication.

```

9734 \bool_if:NTF \l_@@_submatrix_slim_bool
9735 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
9736 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
9737 {
9738   \cs_if_exist:cT
9739   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9740   {
9741     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9742     \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9743     { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9744   }
9745   \cs_if_exist:cT
9746   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9747   {
9748     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9749     \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9750     { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9751   }
9752 }
9753 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
9754 { \@@_impossible_delimiter:n { left } }
9755 {
9756   \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
9757   { \@@_impossible_delimiter:n { right } }
9758   { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
9759 }
9760 \endpgfpicture
9761 }
9762 \group_end:
9763 \ignorespaces
9764 }

```

The argument of the following command will be provided by curryfication.

```

9765 \cs_new_protected:Npn \@@_impossible_delimiter:n #1
9766 {
9767   \bool_if:NTF \l_@@_no_cell_nodes_bool
9768   { \@@_error:n { Impossible~SubMatrix~no~cell~nodes } }
9769   { \@@_error:nn { Impossible~SubMatrix } { #1 } }
9770 }

```

#1 is the left delimiter, **#2** is the right one, **#3** is the subscript and **#4** is the superscript.

```

9771 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
9772 {
9773   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
9774   \dim_set:Nn \l_@@_y_initial_dim
9775   {
9776     \fp_to_dim:n
9777     {
9778       \pgf@y

```

```

9779         + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
9780         - \arrayrulewidth
9781     }
9782 }
9783 \@@_qpoint:n { row - \l_@@_last_i_tl - base }
9784 \dim_set:Nn \l_@@_y_final_dim
9785 {
9786     \fp_to_dim:n
9787     {
9788         \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch
9789         - \arrayrulewidth
9790     }
9791 }
9792 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
9793 {
9794     \cs_if_exist:cT
9795     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
9796     {
9797         \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
9798         \dim_set:Nn \l_@@_y_initial_dim
9799         { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
9800     }
9801     \cs_if_exist:cT
9802     { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
9803     {
9804         \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
9805         \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
9806         { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
9807     }
9808 }
9809 \dim_set:Nn \l_tmpa_dim
9810 {
9811     \l_@@_y_initial_dim - \l_@@_y_final_dim +
9812     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
9813 }
9814 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

9815 \group_begin:
9816 \pgfsetlinewidth { 1.1 \arrayrulewidth }
9817 \@@_set_CTarc:o \l_@@_rules_color_tl
9818 \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`.

```

9819 \seq_map_inline:Nn \g_@@_cols_vlism_seq
9820 {
9821     \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
9822     {
9823         \int_compare:nNnT
9824         { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
9825         {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

9826         \@@_qpoint:n { col - ##1 }
9827         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9828         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9829         \pgfusepathqstroke
9830     }
9831 }
9832 }

```

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.

```

9833 \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
9834 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
9835 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
9836 {
9837   \bool_lazy_and:nnTF
9838   { \int_compare_p:nNn { ##1 } > \c_zero_int }
9839   {
9840     \int_compare_p:nNn
9841     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
9842   {
9843     \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
9844     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9845     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9846     \pgfusepathqstroke
9847   }
9848   { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
9849 }

```

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.

```

9850 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
9851 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
9852 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
9853 {
9854   \bool_lazy_and:nnTF
9855   { \int_compare_p:nNn { ##1 } > \c_zero_int }
9856   {
9857     \int_compare_p:nNn
9858     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
9859   {
9860     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

9861 \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

9862 \dim_set:Nn \l_tmpa_dim
9863 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9864 \str_case:nn { #1 }
9865 {
9866   ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9867   [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
9868   \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9869   }
9870   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

9871 \dim_set:Nn \l_tmpb_dim
9872 { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9873 \str_case:nn { #2 }
9874 {
9875   ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9876   ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
9877   \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9878 }
9879 \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9880 \pgfusepathqstroke
9881 \group_end:
9882 }
9883 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
9884 }

```

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).

```

9885 \str_if_empty:NF \l_@@_submatrix_name_str
9886 {
9887   \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
9888   \l_@@_x_initial_dim \l_@@_y_initial_dim
9889   \l_@@_x_final_dim \l_@@_y_final_dim
9890 }
9891 \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`.

```

9892 \begin { pgfscope }
9893 \pgftransformshift
9894 {
9895   \pgfpoint
9896   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9897   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9898 }
9899 \str_if_empty:NTF \l_@@_submatrix_name_str
9900 { \@@_node_left:nn #1 { } }
9901 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9902 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9903 \pgftransformshift
9904 {
9905   \pgfpoint
9906   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9907   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9908 }
9909 \str_if_empty:NTF \l_@@_submatrix_name_str
9910 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9911 {
9912   \@@_node_right:nnnn #2
9913   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9914 }

```

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`.

```

9915 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9916 \flag_clear_new:N \l_@@_code_flag
9917 \l_@@_code_tl
9918 }

```

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-lj* 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`.

```

9919 \cs_set_eq:NN \@@_old_pgfpointanchor: \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of TikZ nodes which are computed in an expandable way.

The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by currying.

```
9920 \cs_new:Npn \@@_pgfpointanchor:n #1
9921 { \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`).

```
9922 \cs_new:Npn \@@_pgfpointanchor_i:n #1
9923 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }

9924 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9925 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
9926 \str_if_empty:nTF { #1 }
```

First, when the name of the name begins with `\tikz@pp@name`.

```
9927 { \@@_pgfpointanchor_iv:w #2 }
```

And now, when there is no `\tikz@pp@name`.

```
9928 { \@@_pgfpointanchor_ii:n { #1 } }
9929 }
```

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`).

```
9930 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
9931 { \@@_pgfpointanchor_ii:n { #1 } }
```

With the command `\@@_pgfpointanchor_ii:n`, we deal with the actual name of the node (without the `\tikz@pp@name`). First, we have to detect whether it is of the form `i` or 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`.

```
9932 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }
```

```
9933 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
9934 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
9935 \str_if_empty:nTF { #2 }
```

First the case where the argument does *not* contain an hyphen.

```
9936 { \@@_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`).

```
9937 { \@@_pgfpointanchor_iii:w { #1 } #2 }
9938 }
```

The following function is for the case when the name contains an hyphen.

```
9939 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9940 {
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
9941 \@@_env:
9942 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9943 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9944 }
```

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.

```

9945 \tl_const:Nn \c_@@_integers_alist_tl
9946 {
9947   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9948   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9949   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9950   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9951 }

9952 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9953 {

```

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`.

```

9954 \str_case:nVTF { #1 } \c_@@_integers_alist_tl
9955 {
9956   \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9957 \@@_env: -
9958 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9959 { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9960 { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9961 }
9962 {
9963   \str_if_eq:eeTF { #1 } { last }
9964   {
9965     \flag_raise:N \l_@@_code_flag
9966     \@@_env: -
9967     \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9968     { \int_eval:n { \l_@@_last_i_tl + 1 } }
9969     { \int_eval:n { \l_@@_last_j_tl + 1 } }
9970   }
9971   { #1 }
9972 }
9973 }

```

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`).

```

9974 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9975 {
9976   \pgfnode
9977   { rectangle }
9978   { east }
9979   {
9980     \nullfont
9981     $ % $
9982     \@@_color:o \l_@@_delimiters_color_tl
9983     \left #1
9984     \vcenter
9985     {
9986       \nullfont
9987       \hrule \@height \l_tmpa_dim
9988       \@depth \c_zero_dim

```

```

9989             \@width \c_zero_dim
9990         }
9991         \right .
9992         $ % $
9993     }
9994     { #2 }
9995     { }
9996 }

```

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.

```

9997 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9998 {
9999     \pgfnode
10000     { rectangle }
10001     { west }
10002     {
10003         \nullfont
10004         $ % $
10005         \colorlet { current-color } { . }
10006         \@@_color:o \l_@@_delimiters_color_tl
10007         \left .
10008         \vcenter
10009         {
10010             \nullfont
10011             \hrule \@height \l_tmpa_dim
10012                 \@depth \c_zero_dim
10013                 \@width \c_zero_dim
10014         }
10015         \right #1
10016         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
10017         ^ { \color { current-color } \smash { #4 } }
10018         $ % $
10019     }
10020     { #2 }
10021     { }
10022 }

```

33 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

10023 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
10024 {
10025     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
10026     \ignorespaces
10027 }
10028 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
10029 {
10030     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
10031     \ignorespaces
10032 }
10033 \keys_define:nn { nicematrix / Brace }
10034 {
10035     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
10036     left-shorten .value_forbidden:n = true ,
10037     right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,

```

```

10038 right-shorten .value_forbidden:n = true ,
10039 shorten .meta:n = { left-shorten , right-shorten } ,
10040 shorten .value_forbidden:n = true ,
10041 yshift .dim_set:N = \l_@@_brace_yshift_dim ,
10042 color .tl_set:N = \l_tmpa_tl ,
10043 color .value_required:n = true ,
10044 unknown .code:n =
10045     \@@_unknown_key:nn
10046     { nicematrix / Brace }
10047     { Unknown-key-for-Brace }
10048 }

```

#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`.

```

10049 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
10050 {
10051     \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

10052     \@@_compute_i_j:nn { #1 } { #2 }
10053     \bool_lazy_or:nnTF
10054         { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
10055         { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
10056     {
10057         \str_if_eq:eeTF { #5 } { under }
10058         { \@@_error:nn { Construct-too-large } { \UnderBrace } }
10059         { \@@_error:nn { Construct-too-large } { \OverBrace } }
10060     }
10061     {
10062         \tl_clear:N \l_tmpa_tl
10063         \keys_set:nn { nicematrix / Brace } { #4 }
10064         \tl_if_empty:NF \l_tmpa_tl { \color \l_tmpa_tl }
10065         \pgfpicture
10066         \pgfrememberpicturerepositiononpagetrue
10067         \pgf@relevantforpicturesizefalse
10068         \bool_if:NT \l_@@_brace_left_shorten_bool
10069         {
10070             \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
10071             \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
10072             {
10073                 \cs_if_exist:cT
10074                     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
10075                 {
10076                     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
10077
10078                     \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
10079                     { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
10080                 }
10081             }
10082         }
10083         \bool_lazy_or:nnT
10084             { \bool_not_p:n \l_@@_brace_left_shorten_bool }
10085             { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
10086         {
10087             \@@_qpoint:n { col - \l_@@_first_j_tl }
10088             \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
10089         }
10090         \bool_if:NT \l_@@_brace_right_shorten_bool
10091         {
10092             \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
10093             \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
10094             {
10095                 \cs_if_exist:cT

```

```

10096         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
10097         {
10098             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
10099             \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
10100             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
10101         }
10102     }
10103 }
10104 \bool_lazy_or:nnT
10105 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
10106 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
10107 {
10108     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
10109     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
10110 }
10111 \pgfset { inner~sep = \c_zero_dim }
10112 \str_if_eq:eeTF { #5 } { under }
10113 { \@@_underbrace_i:n { #3 } }
10114 { \@@_overbrace_i:n { #3 } }
10115 \endpgfpicture
10116 }
10117 \group_end:
10118 }

```

The argument is the text to put above the brace.

```

10119 \cs_new_protected:Npn \@@_overbrace_i:n #1
10120 {
10121     \@@_qpoint:n { row - \l_@@_first_i_tl }
10122     \pgftransformshift
10123     {
10124         \pgfpoint
10125         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
10126         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
10127     }
10128     \pgfnode
10129     { rectangle }
10130     { south }
10131     {
10132         \vtop
10133         {
10134             \group_begin:
10135             \everycr { }
10136             \halign
10137             {
10138                 \hfil ## \hfil \crcr
10139                 \bool_if:NTF \l_@@_tabular_bool
10140                 { \begin { tabular } { c } #1 \end { tabular } }
10141                 { $ \begin { array } { c } #1 \end { array } $ }
10142                 \cr
10143                 $ % $
10144                 \overbrace
10145                 {
10146                     \hbox_to_wd:nn
10147                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
10148                     { }
10149                 }
10150                 $ % $
10151                 \cr
10152             }
10153             \group_end:
10154         }
10155     }
10156 }
10157 { }

```

```
10158 }
```

The argument is the text to put under the brace.

```
10159 \cs_new_protected:Npn \@@_underbrace_i:n #1
10160 {
10161   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
10162   \pgftransformshift
10163   {
10164     \pgfpoint
10165     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
10166     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
10167   }
10168   \pgfnode
10169   { rectangle }
10170   { north }
10171   {
10172     \group_begin:
10173     \everycr { }
10174     \vbox
10175     {
10176       \halign
10177       {
10178         \hfil ## \hfil \crcr
10179         $ % $
10180         \underbrace
10181         {
10182           \hbox_to_wd:nn
10183           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
10184           { }
10185         }
10186         $ % $
10187         \cr
10188         \bool_if:NTF \l_@@_tabular_bool
10189         { \begin { tabular } { c } #1 \end { tabular } }
10190         { $ \begin { array } { c } #1 \end { array } $ }
10191         \cr
10192       }
10193     }
10194     \group_end:
10195   }
10196   { }
10197   { }
10198 }
```

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`).

```
10199 \AddToHook { package / tikz / after }
10200 {
10201   \tikzset
10202   {
10203     nicematrix / brace / .style =
10204     {
```

```

10205         decoration = { brace , raise = -0.15 em } ,
10206         decorate ,
10207     } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

10208     nicematrix / mirrored-brace / .style =
10209     {
10210         nicematrix / brace ,
10211         decoration = mirror ,
10212     }
10213 }
10214 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

10215 \keys_define:nn { nicematrix / Hbrace }
10216 {
10217     color .code:n = ,
10218     horizontal-label .code:n = ,
10219     horizontal-labels .code:n = ,
10220     shorten .code:n = ,
10221     shorten-start .code:n = ,
10222     shorten-end .code:n = ,
10223     shorten+ .code:n = ,
10224     shorten-start+ .code:n = ,
10225     shorten-end+ .code:n = ,
10226     shorten~+ .code:n = ,
10227     shorten-start~+ .code:n = ,
10228     shorten-end~+ .code:n = ,
10229     brace-shift .code:n = ,
10230     brace-shift+ .code:n = ,
10231     brace-shift~+ .code:n = ,
10232     unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
10233 }

```

Here we need an “fully expandable” command.

```

10234 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
10235 {
10236     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
10237     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
10238     { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
10239 }

```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

10240 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
10241 {
10242     \int_compare:nNnTF \c@iRow < { 2 }
10243     {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

10244     \str_if_eq:nnTF { #2 } { * }
10245     {
10246         \bool_set_true:N \l_@@_nullify_dots_bool
10247         \Ldots
10248         [
10249             line-style = nicematrix / brace ,
10250             #1 ,
10251             up =
10252             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10253         ]
10254     }

```

```

10255     {
10256         \Hdotsfor
10257         [
10258             line-style = nicematrix / brace ,
10259             #1 ,
10260             up =
10261             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10262         ]
10263         { #2 }
10264     }
10265 }
10266 {
10267     \str_if_eq:nnTF { #2 } { * }
10268     {
10269         \bool_set_true:N \l_@@_nullify_dots_bool
10270         \Ldots
10271         [
10272             line-style = nicematrix / mirrored-brace ,
10273             #1 ,
10274             down =
10275             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10276         ]
10277     }
10278     {
10279         \Hdotsfor
10280         [
10281             line-style = nicematrix / mirrored-brace ,
10282             #1 ,
10283             down =
10284             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10285         ]
10286         { #2 }
10287     }
10288 }
10289 \keys_set:nn { nicematrix / Hbrace } { #1 }
10290 }

```

```

10291 \NewDocumentCommand { \@@_Vbrace } { 0 { } m m }
10292 {
10293     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
10294     { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
10295     { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
10296 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

10297 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
10298 {
10299     \int_compare:nNnTF \c@jCol < { 2 }
10300     {
10301         \str_if_eq:nnTF { #2 } { * }
10302         {
10303             \bool_set_true:N \l_@@_nullify_dots_bool
10304             \Vdots
10305             [
10306                 Vbrace ,
10307                 line-style = nicematrix / mirrored-brace ,
10308                 #1 ,
10309                 down =
10310                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10311             ]
10312         }
10313     }

```

```

10314         \Vdotsfor
10315         [
10316             Vbrace ,
10317             line-style = nicematrix / mirrored-brace ,
10318             #1 ,
10319             down =
10320                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10321         ]
10322     { #2 }
10323 }
10324 }
10325 {
10326     \str_if_eq:nnTF { #2 } { * }
10327     {
10328         \bool_set_true:N \l_@@_nullify_dots_bool
10329         \Vdots
10330         [
10331             Vbrace ,
10332             line-style = nicematrix / brace ,
10333             #1 ,
10334             up =
10335                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10336         ]
10337     }
10338     {
10339         \Vdotsfor
10340         [
10341             Vbrace ,
10342             line-style = nicematrix / brace ,
10343             #1 ,
10344             up =
10345                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10346         ]
10347         { #2 }
10348     }
10349 }
10350 \keys_set:nn { nicematrix / Hbrace } { #1 }
10351 }

```

35 The command TikzEveryCell

```

10352 \bool_new:N \l_@@_not_empty_bool
10353 \bool_new:N \l_@@_empty_bool
10354
10355 \keys_define:nn { nicematrix / TikzEveryCell }
10356 {
10357     not-empty .code:n =
10358         \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
10359         { \bool_set_true:N \l_@@_not_empty_bool }
10360         { \@@_error:n { detection-of-empty-cells } } ,
10361     not-empty .value_forbidden:n = true ,
10362     empty .code:n =
10363         \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
10364         { \bool_set_true:N \l_@@_empty_bool }
10365         { \@@_error:n { detection-of-empty-cells } } ,
10366     empty .value_forbidden:n = true ,
10367     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
10368 }
10369

```

```

10370
10371 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
10372 {
10373   \IfPackageLoadedTF { tikz }
10374   {
10375     \group_begin:
10376     \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.

```

10377     \tl_set:Nn \l_tmpa_tl { { #2 } }
10378     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
10379     { \@@_for_a_block:nnnnn #1 }
10380     \@@_all_the_cells:
10381     \group_end:
10382   }
10383   { \@@_error:n { TikzEveryCell-without~tikz } }
10384 }
10385
10386
10387 \cs_new_protected:Nn \@@_all_the_cells:
10388 {
10389   \int_step_inline:nn \c@iRow
10390   {
10391     \int_step_inline:nn \c@jCol
10392     {
10393       \cs_if_exist:cF { cell - ##1 - #####1 }
10394       {
10395         \@@_if_in_corner:nF { ##1 - #####1 }
10396         {
10397           \bool_set_false:N \l_tmpa_bool
10398           \cs_if_exist:cTF
10399           { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
10400           {
10401             \bool_if:NF \l_@@_empty_bool
10402             { \bool_set_true:N \l_tmpa_bool }
10403           }
10404           {
10405             \bool_if:NF \l_@@_not_empty_bool
10406             { \bool_set_true:N \l_tmpa_bool }
10407           }
10408           \bool_if:NT \l_tmpa_bool
10409           {
10410             \@@_block_tikz:nnnnn
10411             \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
10412           }
10413         }
10414       }
10415     }
10416   }
10417 }
10418
10419 \cs_new_protected:Nn \@@_for_a_block:nnnnn
10420 {
10421   \bool_if:NF \l_@@_empty_bool
10422   {
10423     \@@_block_tikz:nnnnn
10424     \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
10425   }
10426   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
10427 }
10428
10429 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
10430 {

```

```

10431 \int_step_inline:nnn { #1 } { #3 }
10432 {
10433     \int_step_inline:nnn { #2 } { #4 }
10434     { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
10435 }
10436 }

```

36 The key draw-trees-in-col

```

10437 \cs_new_protected:Npn \@@_draw_trees:
10438 {
10439     \bool_if:NTF \l_@@_no_cell_nodes_bool
10440     { \@@_error:nn { draw-trees~with~no~cell~nodes } }
10441     \@@_draw_trees_i:
10442 }
10443 \cs_new_protected:Npn \@@_draw_trees_i:
10444 {
10445     \@@_expand_clist_hvlines:NN \g_@@_col_with_trees_clist \c@jCol
10446     \dim_zero_new:N \l_@@_em_dim
10447     \dim_set:Nn \l_@@_em_dim { 1 em }
10448     \dim_zero_new:N \l_@@_ex_dim
10449     \dim_set:Nn \l_@@_ex_dim { 1 ex }
10450     \pgfpicture
10451     \pgfrememberpicturepositiononpagetrue
10452     \pgf@relevantforpicturesizefalse
10453     \dim_compare:nNnT \l_@@_trees_line_width_dim > \c_zero_dim
10454     { \pgfsetlinewidth { \l_@@_trees_line_width_dim } }
10455     \@@_color:o \l_@@_trees_color_tl
10456     \pgfsetcornersarced
10457     { \pgfpoint \l_@@_trees_rounded_corners_dim \l_@@_trees_rounded_corners_dim }
10458     \clist_map_function:NN \g_@@_col_with_trees_clist
10459     \@@_draw_trees_in_col:n
10460     \clist_gclear:N \g_@@_col_with_trees_clist
10461     \endpgfpicture
10462 }

```

```

10463 \cs_new_protected:Npn \@@_draw_trees_in_col:n #1
10464 {

```

The argument is provided by curryfication.

```

10465     \int_compare:nNnTF { #1 } > \c@jCol
10466     { \@@_error:nn { Col~outside~tabular~in~trees } }
10467     {
10468         \int_compare:nNnTF { #1 } = \c@jCol
10469         { \@@_error:nn { Last~col~in~trees } }
10470         \@@_draw_trees_in_col_i:n
10471     }
10472     { #1 }
10473 }

```

```

10474 \cs_new_protected:Npn \@@_draw_trees_in_col_i:n #1
10475 {
10476     \int_set:Nn \l_tmpa_int { 1 }
10477     \int_step_inline:nn { \c@iRow + 1 }
10478     {
10479         \cs_if_exist:cT
10480         { pgf @ sh @ ns @ \@@_env: - ##1 - #1 }
10481         {
10482             \int_compare:nNnT { ##1 } > { \l_tmpa_int + 1 }
10483             {

```

Now, you will, potentially, draw a tree.

```

10484         \@@_draw_tree:nee
10485         { #1 }
10486         { \int_use:N \l_tmpa_int }
10487         { \int_eval:n { ##1 - 1 } }
10488     }
10489     \int_set:Nn \l_tmpa_int { ##1 }
10490 }
10491 }
10492 \int_compare:nNnT \c@iRow > \l_tmpa_int
10493 {
10494     \@@_draw_tree:nee
10495     { #1 }
10496     { \int_use:N \l_tmpa_int }
10497     { \int_eval:n { \c@iRow } }
10498 }
10499 }

```

#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.

```

10500 \cs_new_protected:Npn \@@_draw_tree:nnn #1 #2 #3
10501 {

```

\l_tmpa_dim will be the x -value of the vertical rule that we will draw.

```

10502     \pgfpointanchor { \@@_env: - #1 } { 5 }
10503     \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.

```

10504     \bool_set_false:N \l_tmpa_bool
10505     \int_step_inline:nnnn { #3 } { -1 } { #2 + 1 }
10506     {
10507         \cs_if_exist:cT
10508         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #1 + 1 } }

```

We have found the last branch to draw.

```

10509     {
10510         \pgfpointanchor
10511         { \@@_env: - ##1 - \int_eval:n { #1 + 1 } }
10512         { base-west }
10513         \pgfpathmoveto
10514         {
10515             \pgfpoint
10516             { \dim_eval:n { \pgf@x - 0.7 \l_@@_ex_dim } }
10517             { \dim_eval:n { \pgf@y + 0.25 \l_@@_em_dim } }
10518         }
10519         \pgfpathlineto { \pgfpoint \l_tmpa_dim \pgf@y }
10520         \bool_if:NF \l_tmpa_bool
10521         {
10522             \pgfpointanchor{ \@@_env: - #2 - #1 } { south }
10523             \pgfpathlineto
10524             { \pgfpoint \l_tmpa_dim { \dim_eval:n { \pgf@y - 3pt } } }
10525             \bool_set_true:N \l_tmpa_bool
10526         }
10527         \pgfusepath { stroke }
10528     }
10529 }
10530 }
10531 \cs_generate_variant:Nn \@@_draw_tree:nnn { n e e }

```

37 The key create-blocks-in-col

```

10532 \cs_new_protected:Npn \@@_create_blocks_in_col:
10533 {

```

```

10534 \@@_expand_clist_hvlines:NN \g_@@_cbic_clist \c@jCol
10535 \clist_map_inline:Nn \g_@@_cbic_clist
10536 {
10537   \cs_set:cpn
10538   { pgf @ sh @ ns @ \@@_env: - \int_eval:n { \c@iRow + 1 } - ##1 }
10539   { rien }
\l_tmpa_int will be the first row of the block being created.
10540 \int_set:Nn \l_tmpa_int { 1 }
10541 \int_step_inline:nn { \c@iRow + 1 }
10542 {
10543   \bool_set_false:N \l_tmpa_bool
10544   \cs_if_exist:cT { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 }
10545   { \bool_set_true:N \l_tmpa_bool }
10546   \clist_if_in:NnT \g_@@_false_rows_clist { #####1 }
10547   { \bool_set_true:N \l_tmpa_bool }
10548   \bool_if:NT \l_tmpa_bool
10549   {
10550     \int_compare:nNnT { #####1 } > { \l_tmpa_int + 1 }
10551     {
10552       \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
10553       {
10554         { \int_use:N \l_tmpa_int }
10555         { ##1 }
10556         { \int_eval:n { #####1 - 1 } }
10557         { ##1 }
10558         { }
10559       }
10560     }
10561     \int_set:Nn \l_tmpa_int { #####1 }
10562   }
10563 }
10564 }
10565 \clist_gclear:N \g_@@_cbic_clist
10566 }

```

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.

```

10567 \cs_new_protected:Npn \@@_ShowCellNamesCodeBefore
10568 { \tl_put_right:Nn \@@_actually_color: \@@_ShowCellNames } % noqa

10569 \NewDocumentCommand \@@_ShowCellNames { }
10570 {
10571   \bool_if:NT \l_@@_in_code_after_bool
10572   {
10573     \pgfpicture
10574     \pgfrememberpicturerepositiononpagetrue
10575     \pgf@relevantforpicturesizefalse
10576     \pgfpathrectanglecorners
10577     { \@@_qpoint:n { 1 } }
10578     { \@@_qpoint:n { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } } }
10579     \pgfsetfillopacity { 0.75 }
10580     \pgfsetfillcolor { white }
10581     \pgfusepathqfill
10582   }
10583 }

```

```

10584 \dim_gzero_new:N \g_@@_tmpc_dim
10585 \dim_gzero_new:N \g_@@_tmpd_dim
10586 \dim_gzero_new:N \g_@@_tmpe_dim
10587 \int_step_inline:nn \c@iRow
10588 {
10589   \clist_if_in:NnF \g_@@_false_rows_clist { ##1 }
10590   { \@@_show_cell_names_one_row:n { ##1 } }
10591 }
10592 }

10593 \cs_new_protected:Npn \@@_show_cell_names_one_row:n #1
10594 {
10595   \bool_if:NTF \l_@@_in_code_after_bool
10596   {
10597     \pgfpicture
10598     \pgfrememberpicturepositiononpagetrue
10599     \pgf@relevantforpicturesizefalse
10600   }
10601   { \begin { pgfpicture } }
10602   \@@_qpoint:n { row - #1 }
10603   \dim_set_eq:NN \l_tmpa_dim \pgf@y
10604   \@@_qpoint:n { row - \int_eval:n { #1 + 1 } }
10605   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
10606   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
10607   \bool_if:NTF \l_@@_in_code_after_bool
10608   { \endpgfpicture }
10609   { \end { pgfpicture } }
10610   \int_step_tokens:nn \c@jCol
10611   { \@@_show_cell_names_one_cell:nn { #1 } }
10612 }

10613 \cs_new_protected:Npn \@@_show_cell_names_one_cell:nn #1 #2
10614 {
10615   \clist_if_in:NnF \g_@@_false_cols_clist { #2 }
10616   { \@@_show_cell_names_one_cell_i:nn { #1 } { #2 } }
10617 }

10618 \cs_new_protected:Npn \@@_show_cell_names_one_cell_i:nn #1 #2
10619 {
10620   \hbox_set:Nn \l_tmpa_box
10621   {
10622     \normalfont \Large \sffamily \bfseries
10623     \bool_if:NTF \l_@@_in_code_after_bool
10624     { \color { red } }
10625     { \color { red ! 50 } }
10626     #1 - #2
10627   }
10628   \bool_if:NTF \l_@@_in_code_after_bool
10629   {
10630     \pgfpicture
10631     \pgfrememberpicturepositiononpagetrue
10632     \pgf@relevantforpicturesizefalse
10633   }
10634   { \begin { pgfpicture } }
10635   \@@_qpoint:n { col - #2 }
10636   \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
10637   \@@_qpoint:n { col - \int_eval:n { #2 + 1 } }
10638   \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
10639   \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
10640   \bool_if:NTF \l_@@_in_code_after_bool
10641   { \endpgfpicture }
10642   { \end { pgfpicture } }
10643   \fp_set:Nn \l_tmpa_fp
10644   {

```

```

10645     \fp_min:nn
10646     {
10647         \fp_min:nn
10648         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
10649         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
10650     }
10651     { 1.0 }
10652 }
10653 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
10654 \pgfpicture
10655 \pgfrememberpicturepositiononpagetrue
10656 \pgf@relevantforpicturesizefalse
10657 \pgftransformshift
10658 {
10659     \pgfpoint
10660     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
10661     \g_tmpa_dim
10662 }
10663 \pgfnode
10664 { rectangle }
10665 { center }
10666 { \box_use:N \l_tmpa_box }
10667 { }
10668 { }
10669 \endpgfpicture
10670 }

```

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.

```

10671 \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.

```

10672 \bool_new:N \g_@@_footnote_bool
10673 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
10674 {
10675     You-have-used-the-key~' \l_keys_key_str '~when-loading-nicematrix~
10676     but-that-key-is-unknown. \\
10677     It-will-be-ignored. \\
10678     For-a-list-of-the-available-keys,~type-H-<return>.
10679 }
10680 {
10681     The-available-keys-are~(in~alphabetic~order):~
10682     footnote,~
10683     footnotehyper,~
10684     messages-for-Overleaf,~
10685     renew-dots-and~
10686     renew-matrix.
10687 }

```

```

10688 \keys_define:nn { nicematrix }
10689 {
10690   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
10691   renew-dots .value_forbidden:n = true ,
10692   renew-matrix .code:n = \@@_renew_matrix: ,
10693   renew-matrix .value_forbidden:n = true ,
10694   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
10695   footnote .bool_set:N = \g_@@_footnote_bool ,
10696   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
10697   unknown .code:n = \@@_error:n { Unknown~key~for~package }
10698 }
10699 \ProcessKeyOptions

10700 \@@_msg_new:nn { footnote~with~footnotehyper~package }
10701 {
10702   You~can't~use~the~option~'footnote'~because~the~package~
10703   footnotehyper~has~already~been~loaded.~
10704   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
10705   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10706   of~the~package~footnotehyper.\\
10707   The~package~footnote~won't~be~loaded.
10708 }

10709 \@@_msg_new:nn { footnotehyper~with~footnote~package }
10710 {
10711   You~can't~use~the~option~'footnotehyper'~because~the~package~
10712   footnote~has~already~been~loaded.~
10713   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
10714   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10715   of~the~package~footnote.\\
10716   The~package~footnotehyper~won't~be~loaded.
10717 }

10718 \bool_if:NT \g_@@_footnote_bool
10719 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10720 \IfClassLoadedTF { beamer }
10721 { \bool_set_false:N \g_@@_footnote_bool }
10722 {
10723   \IfPackageLoadedTF { footnotehyper }
10724   { \@@_error:n { footnote~with~footnotehyper~package } }
10725   { \usepackage { footnote } }
10726 }
10727 }

10728 \bool_if:NT \g_@@_footnotehyper_bool
10729 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10730 \IfClassLoadedTF { beamer }
10731 { \bool_set_false:N \g_@@_footnote_bool }
10732 {
10733   \IfPackageLoadedTF { footnote }
10734   { \@@_error:n { footnotehyper~with~footnote~package } }
10735   { \usepackage { footnotehyper } }
10736 }
10737 \bool_set_true:N \g_@@_footnote_bool
10738 }

```

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.

```
10739 \IfPackageLoadedF { underscore }
10740 {
10741   \AddToHook { package / underscore / after }
10742     { \@@_error:n { underscore~after~nicematrix } }
10743 }
```

41 Compatibility with threeparttable

```
10744 \AddToHook { package / threeparttable / after }
10745 {
10746   \AddToHook { env / threeparttable / begin }
10747     {
10748       \TPT@hookin { NiceTabular }
10749       \TPT@hookin { NiceTabular* }
10750       \TPT@hookin { NiceTabularX }
10751     }
10752 }
```

42 Gestion of the errors

42.1 Security in the CodeBefore and the CodeAfter

```
10753 \cs_new_protected:Npn \@@_security_font_commands:
10754 {
10755   \cs_set_eq:NN \@@_old_small: \small
10756   \cs_set_eq:NN \@@_old_footnotesize: \footnotesize
10757   \cs_set_eq:NN \@@_old_scriptsize: \scriptsize
10758   \cs_set_eq:NN \@@_old_tiny: \tiny
10759   \cs_set_eq:NN \small \@@_small:
10760   \cs_set_eq:NN \footnotesize \@@_footnotesize:
10761   \cs_set_eq:NN \scriptsize \@@_scriptsize:
10762   \cs_set_eq:NN \tiny \@@_tiny:
10763   \everyhbox
10764   {
10765     \cs_set_eq:NN \small \@@_old_small:
10766     \cs_set_eq:NN \footnotesize \@@_old_footnotesize:
10767     \cs_set_eq:NN \scriptsize \@@_old_scriptsize:
10768     \cs_set_eq:NN \tiny \@@_old_tiny:
10769   }
10770 }

10771 \cs_set_protected:Npn \@@_small: { \@@_font_command_error:N \small }
10772 \cs_set_protected:Npn \@@_footnotsize: { \@@_font_command_error:N \footnotesize }
10773 \cs_set_protected:Npn \@@_scriptsize: { \@@_font_command_error:N \scriptsize }
10774 \cs_set_protected:Npn \@@_tiny: { \@@_font_command_error:N \tiny }

10775 \cs_new_protected:Npn \@@_font_command_error:N #1
10776 { \@@_error:nn { font-command } { #1 } }

10777 \@@_msg_new:nn { font-command }
10778 {
10779   Font~command~not~allowed.\\
10780   You~can't~use~the~command~#1~directly~in~the~
```

```

10781 \bool_if:NTF \l_@@_in_code_after_bool
10782 { \token_to_str:N \CodeAfter}
10783 { \token_to_str:N \CodeBefore}.~Your~command~will~be~ignored.
10784 }

```

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.

```

10785 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
10786 {
10787   \str_if_eq:eeTF
10788   { \str_item:Nn \l_keys_key_str { \str_count:N \l_keys_key_str } }
10789   { + }
10790   {
10791     \str_set:Ne \l_tmpa_str
10792     { \str_range:Nnn \l_keys_key_str { 1 } { \str_count:N \l_keys_key_str - 1 } }
10793     \bool_set_false:N \l_tmpa_bool
10794     \clist_map_inline:nn { #1 }
10795     {
10796       \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10797       {
10798         \@@_error:n { key~without~+~exists }
10799         \bool_set_true:N \l_tmpa_bool
10800         \clist_map_break:
10801       }
10802     }
10803     \bool_if:NF \l_tmpa_bool
10804     {
10805       \str_set:Ne \l_keys_key_str { \tl_trim_right_spaces:V \l_tmpa_str }
10806       \@@_unknown_key_i:nn { #1 } { #2 }
10807     }
10808   }
10809   { \@@_unknown_key_i:nn { #1 } { #2 } }
10810 }

```

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.

```

10811 \cs_new_protected:Npn \@@_unknown_key_i:nn #1 #2
10812 {
10813   \str_set_eq:NN \l_tmpa_str \l_keys_key_str
10814   \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
10815   \str_set:Ne \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
10816   \bool_set_false:N \l_tmpa_bool
10817   \clist_map_inline:nn { #1 }
10818   {
10819     \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10820     {
10821       \@@_error:n { key~with~normal~form~exists }
10822       \bool_set_true:N \l_tmpa_bool
10823       \clist_map_break:
10824     }
10825   }
10826   \bool_if:NF \l_tmpa_bool
10827   {
10828     \@@_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).

```

10829     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10830     { \msg_info:nn { nicematrix } { #2~+ } }
10831   }
10832 }
10833 \@@_msg_new:nn { key~without~+~exists }
10834 {
10835   Problem~with~a~key.\\
10836   The~key~'\tl_trim_right_spaces:V \l_tmpa_str'~exists~but~does~not~accept~an~
10837   additive~syntax~(with~+=).~It~will~be~ignored.
10838 }
10839 \@@_msg_new:nn { key~with~normal~form~exists }
10840 {
10841   No~key~'\l_keys_key_str'.\\
10842   It~will~be~ignored.~Maybe~you~want~to~use~the~key~'\l_tmpa_str'.
10843 }
10844 \str_const:Ne \c_@@_available_keys_str
10845 {
10846   \bool_if:nT { ! \g_@@_messages_for_Overleaf_bool }
10847   { For~a~list~of~the~available~keys,~type~H~<return>. }
10848 }
10849 \seq_new:N \g_@@_types_of_matrix_seq
10850 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
10851 {
10852   NiceMatrix ,
10853   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
10854 }
10855 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
10856 { \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`.

```

10857 \cs_new_protected:Npn \@@_err_too_many_cols:
10858 {
10859   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
10860   { \@@_fatal:nn { too-many-cols-for-array } }
10861   \int_compare:nNnT \l_@@_last_col_int = { -2 }
10862   { \@@_fatal:n { too-many-cols-for-matrix } }
10863   \int_compare:nNnT \l_@@_last_col_int = { -1 }
10864   { \@@_fatal:n { too-many-cols-for-matrix } }
10865   \bool_if:NF \l_@@_last_col_without_value_bool
10866   { \@@_fatal:n { too-many-cols-for-matrix-with-last-col } }
10867 }

```

The following command must *not* be protected since it's used in an error message.

```

10868 \cs_new:Npn \@@_message_hdotsfor:
10869 {
10870   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
10871   { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
10872     \token_to_str:N \Hbrace \ is~incorrect. }
10873 }
10874 \cs_new_protected:Npn \@@_Hline_in_cell:
10875 { \@@_error:n { Misuse~of~Hline } }

```

```

10876 \@@_msg_new:nn { Misuse-of-Hline }
10877 {
10878   Misuse-of~\token_to_str:N \Hline. \\
10879   Error-in-the-row~ \int_use:N \c@iRow\ of~your~\@@_full_name_env:~
10880   \token_to_str:N \Hline\ (like~\token_to_str:N \hline)~must-be-used-only~
10881   at~the~beginning-of~a~row.~Maybe-you-have-forgotten-a~\token_to_str:N \\~
10882   Your-command-will-be-ignored.
10883 }

10884 \cs_new_protected:Npn \@@_hdashedline:
10885 { \@@_error:n { Misuse-of-hdashedline } }
10886 \cs_new_protected:Npn \@@_cdashedline:n #1
10887 { \@@_error:n { Misuse-of~cdashedline } }

10888 \@@_msg_new:nn { Misuse-of-hdashedline }
10889 {
10890   You-should-load-TikZ~(with~\token_to_str:N \usepackage \{tikz\})~in-order~
10891   to-be-able-to-use-the-command~\token_to_str:N \hdashedline.~Your-command~
10892   will-be-ignored.
10893 }
10894 \@@_msg_new:nn { Misuse-of-cdashedline }
10895 {
10896   You-should-load-TikZ~(with~\token_to_str:N \usepackage \{tikz\})~in-order~
10897   to-be-able-to-use-the-command~\token_to_str:N \cdashedline.~Your-command~
10898   will-be-ignored.
10899 }

10900 \@@_msg_new:nn { hvlines,~rounded-corners-and-corners }
10901 {
10902   Incompatible-options.\\
10903   You-should-not-use~'hvlines',~'rounded-corners'~and~'corners'~at-the-same-time.~
10904   The-output-will-not-be-reliable.
10905 }

10906 \@@_msg_new:nn { Body-alone }
10907 {
10908   \token_to_str:N \Body\ alone. \\
10909   You-have-used~\token_to_str:N \Body\ without~\token_to_str:N \CodeBefore.
10910 }

10911 \@@_msg_new:nn { Misuse-of~CodeBefore }
10912 {
10913   Misuse-of~\token_to_str:N \CodeBefore. \\
10914   \token_to_str:N \CodeBefore\ must-be-used-only-at-the-beginning-of~
10915   the~environment.
10916 }

10917 \@@_msg_new:nn { NiceTabularX~probably-required }
10918 {
10919   Incorrect-syntax.\\
10920   You~probably~want~to~use~\{NiceTabularX\}~whose~first~argument~is~the~
10921   ~width~that~you~wish~for~your~tabular.~But~you~can~also~use~\{NiceTabular\}~
10922   with~the~key~'width'.
10923 }

10924 \@@_msg_new:nn { cellcolor-in-Block }
10925 {
10926   Misuse-of~\token_to_str:N \cellcolor \\
10927   You-can't-use~\token_to_str:N \cellcolor\ in~\token_to_str:N \Block\
10928   \bool_if:NTF \l_@@_amp_in_blocks_bool
10929     { (but-you-could-use-it-in-a-sub-block-since~'&-in-blocks'~is-in-force) }
10930     { (it's-possible-in-a-sub-block-when~'&-in-blocks'~is-in-force) }
10931   .~Here,~you-should-use-the-key~'fill'~of~the-block.~Your-command-will-be-ignored.
10932 }

10933 \@@_msg_new:nn { rowcolor-in-Block }
10934 {

```

```

10935 Misuse-of~\token_to_str:N \rowcolor \\
10936 You-can't-use~\token_to_str:N \rowcolor\ in~\token_to_str:N \Block.~
10937 However,~it's-possible-to-color-the-block-with-its-key~'fill'.~
10938 Your-command-will-be-ignored.
10939 }
10940 \@@_msg_new:nn { key-color-inside }
10941 {
10942 Deleted-key.\\
10943 The-key~'color-inside'~(and-its-alias~'colortbl-like')~has-been-deleted-in
10944 ~'nicematrix'~and-must-not-be-used.
10945 }
10946 \@@_msg_new:nn { Misuse-of-FalseRow }
10947 {
10948 Misuse-of~\token_to_str:N \FalseRow \\
10949 Error-in-the-row~ \int_use:N \c@iRow\ of~your~\@@_full_name_env:~
10950 \token_to_str:N \FalseRow\ must-be-used-only-at-the-beginning-of-a-row.~
10951 Your-command-will-be-ignored.
10952 }
10953 \@@_msg_new:nn { Invalid-argument-for-w }
10954 {
10955 Invalid-argument-for-w.\\
10956 You-have-used-the-type-of~alignment~'#1'~for~your~column~'w'~
10957 but-only~'c',~'r',~'l'~and~'s'~are-allowed-in-a-column~'w'.~
10958 If-you-go-on,~'c'~will-be-used.
10959 }
10960 \@@_msg_new:nn { invalid-weight }
10961 {
10962 Unknown-key.\\
10963 The-key~'\l_keys_key_str'~of~your~column~X~is-unknown-and-will-be-ignored.~
10964 The-available-keys-are:~l,~c,~r,~t~(=p),~m,~b,~V~
10965 \IfPackageLoadedTF { varwidth }
10966 { (since~'varwidth'~is-loaded)~}
10967 { (if-you-load~'varwidth')~}
10968 and-real-numbers-for~the~weight-of~the~X~column.
10969 }
10970 \@@_msg_new:nn { last-col-not-used }
10971 {
10972 Column-not-used.\\
10973 The-key~'last-col'~is-in-force-but-you-have-not-used-that~last~column~
10974 in~your~\@@_full_name_env: .~However,~you-can-go-on.
10975 }
10976 \@@_msg_new:nn { too-many-cols-for-matrix-with-last-col }
10977 {
10978 Too-many-columns.\\
10979 In~the~row~ \int_eval:n { \c@iRow },~
10980 you-try-to-use-more-columns~
10981 than-allowed-by~your~ \@@_full_name_env: .
10982 \@@_message_hdotsfor: \
10983 The-maximal-number-of~columns-is~ \int_eval:n { \l_@@_last_col_int - 1 }~
10984 (plus~the~exterior~columns).~
10985 Maybe-you-have-forgotten-a~\token_to_str:N \\
10986 }
10987 \@@_msg_new:nn { too-many-cols-for-matrix }
10988 {
10989 Too-many-columns.\\
10990 In~the~row~ \int_eval:n { \c@iRow },~
10991 you-try-to-use-more-columns-than-allowed-by~your~ \@@_full_name_env: .
10992 \@@_message_hdotsfor: \
10993 Recall~that~the~maximal-number-of~columns-for~a~matrix~
10994 (excepted~the~potential~exterior~columns)~is~fixed~by~the~
10995 LaTeX~counter~'MaxMatrixCols'.~

```

```

10996     Its-current-value-is~ \int_use:N \c@MaxMatrixCols \
10997     (use~ \token_to_str:N \setcounter \ to-change-that-value).~
10998     Maybe,~you-have-forgotten-a-\token_to_str:N \\.
10999 }

11000 \cs_set:Npn \@@_message_about_false_cols:
11001 {
11002     \int_compare:nNnTF { \clist_count:N \g_@@_false_cols_clist } = 1
11003     { (including-one~false~column~F)~ }
11004     {
11005         \int_compare:nNnT { \clist_count:N \g_@@_false_cols_clist } > 1
11006         {
11007             (including~\int_to_arabic:n { \clist_count:N \g_@@_false_cols_clist }~
11008             false~columns~F)~
11009         }
11010     }
11011 }

11012 \@@_msg_new:nn { too-many-cols-for-array }
11013 {
11014     Too-many-columns.\\
11015     In-the-row~ \int_eval:n { \c@iRow } ,~
11016     ~you-try-to-use-more-columns-than-allowed-by-your~
11017     \@@_full_name_env: . \@@_message_hdotsfor: \ The-maximal-number-of-columns-is~
11018     \int_use:N \g_@@_static_num_of_col_int \
11019     \@@_message_about_false_cols:
11020     \bool_if:nT
11021     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
11022     { (plus-the-exterior-ones)~}
11023     since-the-preamble-is~' \g_@@_user_preamble_tl '~
11024     Maybe,~you-have-forgotten-a-\token_to_str:N \\.
11025 }

11026 \@@_msg_new:nn { columns-not-used }
11027 {
11028     Columns-not-used.\\
11029     The-preamble-of-your~ \@@_full_name_env: \ is~
11030     '\exp_not:V \g_@@_user_preamble_tl'.~
11031     It-announces~ \int_use:N \g_@@_static_num_of_col_int \
11032     columns~\@@_message_about_false_cols:
11033     but-you-only-used~ \int_use:N \c@jCol .~The-columns-that-you-did-not~
11034     use-won't-be-created.~You-won't-have-similar-warning~till-the-end-of-the-document.
11035 }

11036 \str_const:Nn \c_@@_explanation_no_cell_nodes_str
11037 {
11038     because-the-key~'no-cell-nodes'~is-in-force~
11039     (you-should-deactivate-the-key~'no-cell-nodes'~whose-only-goal~
11040     is-to-speed-compilation-up)
11041 }

11042 \@@_msg_new:nn { xdots-without-cell-nodes }
11043 {
11044     Command~#1 forbidden.\\
11045     You-can't-use-the-command~#1~\c_@@_explanation_no_cell_nodes_str.~
11046     Your-command-will-be-ignored.
11047 }

11048 \@@_msg_new:nn { Misuse-of-NiceTabularNotes }
11049 {
11050     Misuse-of~\token_to_str:N \NiceTabularNotes \\
11051     \token_to_str:N~\NiceTabularNotes\ should-be-used-only-after-at~tabular~
11052     which-uses~`notes/no-print`.~ Your-command-will-be-ignored.
11053 }

```

```

11054 \@@_msg_new:nn { empty-preamble }
11055 {
11056   Empty~preamble.\\
11057   The~preamble-of~your~ \@@_full_name_env: \ is~empty.
11058 }
11059 \@@_msg_new:nn { in~first~col }
11060 {
11061   Misuse~of~#1\\
11062   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.~
11063   Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'first~col'.
11064 }
11065 \@@_msg_new:nn { in~last~col }
11066 {
11067   Misuse~of~#1\\
11068   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.~
11069   Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'last~col'.
11070 }
11071 \@@_msg_new:nn { in~first~row }
11072 {
11073   Misuse~of~#1\\
11074   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.~
11075   Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'first~row'.
11076 }
11077 \@@_msg_new:nn { in~last~row }
11078 {
11079   Misuse~of~#1\\
11080   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.~
11081   Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'last~row'.
11082 }
11083 \@@_msg_new:nn { TopRule~without~booktabs }
11084 {
11085   Misuse~of~#1\\
11086   You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.~
11087   You~should~load~'booktabs'~(before~or~after~'nicematrix').~
11088   Your~command~will~be~ignored.
11089 }
11090 \@@_msg_new:nn { rotate~in~p~col }
11091 {
11092   \token_to_str:N \rotate\ forbidden.\\
11093   You~should~not~use~\token_to_str:N \rotate\ in~a~column~of~type~'p',~
11094   'b',~'m'\IfPackageLoadedTF { varwidth } { ,~'X'~or~'V' } { ~or~'X' }.~
11095   If~you~go~on,~maybe~you~won't~have~the~expected~output.~You~should~
11096   consider~the~classical~command~\token_to_str:N \rotatebox.
11097 }
11098 \@@_msg_new:nn { Not~empty~cell~in~false~column }
11099 {
11100   Not~empty~cell.\\
11101   The~cell~( #1~#2 )~of~your~\@@_full_name_env: \
11102   is~not~empty~although~the~column~#2~is~a~
11103   "false~column"~(letter~F~in~the~preamble).~
11104   \bool_if:NTF \l_@@_light_syntax_bool
11105     { You~should~put~\{\}~for~an~empty~cell. }
11106     {
11107       \int_compare:nNtT { #2 } > 1
11108       { Recall~that~you~have~to~put~*two*~ampersands~(&&). }
11109     }
11110 }
11111 \@@_msg_new:nn { TopRule~without~tikz }
11112 {
11113   Misuse~of~#1\\

```

```

11114     You~can't~use~the~command~ #1 because~TikZ~is~not~loaded.~
11115     You~should~load~TikZ~(with~\token_to_str:N \usepackage \{tikz\}).~
11116     \IfPackageLoadedF { booktabs }
11117         { You~should~also~load~'booktabs'~
11118           with~\token_to_str:N \usepackage \{booktabs\}.~ }
11119     Your~command~will~be~ignored.
11120 }

11121 \@@_msg_new:nn { caption~outside~float }
11122 {
11123     Key~caption~forbidden.\\
11124     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
11125     environment~(such~as~\{table\}).~Your~key~will~be~ignored.
11126 }

11127 \@@_msg_new:nn { short~caption~without~caption }
11128 {
11129     You~should~not~use~the~key~'short~caption'~without~'caption'.~
11130     However,~your~'short~caption'~will~be~used~as~'caption'.
11131 }

11132 \@@_msg_new:nn { double~closing~delimiter }
11133 {
11134     Double~delimiter.\\
11135     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
11136     delimiter.~This~delimiter~will~be~ignored.~You~can~try~to~use~
11137     \token_to_str:N \SubMatrix\ in~the~\token_to_str:N \CodeAfter.
11138 }

11139 \@@_msg_new:nn { delimiter~after~opening }
11140 {
11141     Double~delimiter.\\
11142     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
11143     delimiter.~That~delimiter~will~be~ignored.
11144 }

11145 \@@_msg_new:nn { bad~option~for~line~style }
11146 {
11147     Bad~line~style.\\
11148     Since~you~haven't~loaded~TikZ,~the~only~value~you~can~give~to~'line~style'~
11149     is~'standard'.~Your~key~will~be~ignored.~You~can~load~TikZ~with~
11150     \token_to_str:N \usepackage \{tikz\},~before~or~after~'nicematrix'.
11151 }

11152 \@@_msg_new:nn { draw~trees~with~no~cell~nodes }
11153 {
11154     Incompatible~keys.\\
11155     You~can't~use~the~key~'draw~trees~in~col'~
11156     here~\c_@@_explanation_no_cell_nodes_str.~
11157     If~you~go~on,~that~key~will~be~ignored.
11158 }

11159 \@@_msg_new:nn { corners~with~no~cell~nodes }
11160 {
11161     Incompatible~keys.\\
11162     You~can't~use~the~key~'corners'~here~\c_@@_explanation_no_cell_nodes_str.~
11163     If~you~go~on,~that~key~will~be~ignored.
11164 }

11165 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
11166 {
11167     Incompatible~keys.\\
11168     You~can't~create~'extra~nodes'~here~\c_@@_explanation_no_cell_nodes_str.~
11169     If~you~go~on,~those~extra~nodes~won't~be~created.
11170 }

11171 \@@_msg_new:nn { Identical~notes~in~caption }
11172 {
11173     Identical~tabular~notes.\\

```

```

11174     You~can't~put~several~notes~with~the~same~content~in~
11175     \token_to_str:N \caption \ (but~it's~possible~in~the~main~tabular).~
11176     If~you~go~on,~the~output~will~probably~be~erroneous.
11177 }

11178 \@@_msg_new:nn { tabularnote~below~the~tabular }
11179 {
11180     \token_to_str:N \tabularnote \ forbidden\\
11181     You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
11182     of~your~tabular~because~the~caption~will~be~composed~below~
11183     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
11184     key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .~
11185     Your~ \token_to_str:N \tabularnote \ will~be~ignored~and~
11186     no~similar~error~will~raised~in~this~document.
11187 }

11188 \@@_msg_new:nn { Unknown~key~for~rules }
11189 {
11190     Unknown~key.\\
11191     There~are~only~three~keys~available~here:~'width',~'color'~and~
11192     'fix~vertex'.~Your~key~' \l_keys_key_str '~will~be~ignored.
11193 }

11194 \@@_msg_new:nn { Unknown~key~for~trees }
11195 {
11196     Unknown~key.\\
11197     There~are~only~three~keys~available~here:~width~color~and~
11198     rounded~corners.~Your~key~' \l_keys_key_str '~will~be~ignored.
11199 }

11200 % \end{macrocode}
11201 %
11202 %
11203 % \begin{macrocode}
11204 \@@_msg_new:nn { Unknown~key~for~Hbrace }
11205 {
11206     Unknown~key.\\
11207     You~have~used~the~key~' \l_keys_key_str '~but~the~only~
11208     keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
11209     and~ \token_to_str:N \Vbrace \ are:~'brace~shift(+)',~'color',~
11210     'horizontal~label(s)',~'shorten'~'shorten~end'~
11211     and~'shorten~start'.
11212 }

11213 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
11214 {
11215     Unknown~key.\\
11216     There~is~only~two~keys~available~here:~
11217     'empty'~and~'not~empty'.~Your~key~' \l_keys_key_str '~will~be~ignored.
11218 }

11219 \@@_msg_new:nn { Unknown~key~for~rotate }
11220 {
11221     Unknown~key.\\
11222     The~only~keys~available~here~are~'c'~and~'-90'.~
11223     Your~key~' \l_keys_key_str '~will~be~ignored.
11224 }

11225 \@@_msg_new:nnn { Unknown~key~for~custom~line }
11226 {
11227     Unknown~key.\\
11228     The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
11229     It~you~go~on,~you~will~probably~have~other~errors. \\
11230     \c_@@_available_keys_str
11231 }
11232 {
11233     The~available~keys~are~(in~alphabetic~order):~
11234     ccommand,~

```

```

11235     color,~
11236     dashed \IfPackageLoadedF { tikz } { ~ (when-TikZ-is-loaded)} ,~
11237     command,~
11238     dotted,~
11239     letter,~
11240     multiplicity,~
11241     sep-color,~
11242     tikz \IfPackageLoadedF { tikz } { ~ (when-TikZ-is-loaded)},~
11243     and-total-width.
11244 }

11245 \@@_msg_new:nnn { Unknown-key-for-default-line }
11246 {
11247     Unknown-key.\
11248     The-key~' \l_keys_key_str '~is-unknown-in-a~'default-line'.~
11249     It-you-go-on,~you-will-probably-have-other-errors. \
11250     \c_@@_available_keys_str
11251 }
11252 {
11253     The-available-keys-are~(in-alphabetic-order):~
11254     color,~
11255     dashed \IfPackageLoadedF { tikz } { ~ (when-TikZ-is-loaded)} ,~
11256     dotted,~
11257     multiplicity,~
11258     sep-color,~
11259     tikz~\IfPackageLoadedF { tikz } { (when-TikZ-is-loaded)~}
11260     and-total-width.
11261 }

11262 \@@_msg_new:nn { dashed-for-default-line-without-TikZ}
11263 {
11264     Key~'dashed'~forbidden.\
11265     If-you-want-to-use-the-value~'dashed'~for~'default-line',~
11266     you-must-load-TikZ~(with~\token_to_str:N \usepackage \{tikz\}).
11267 }

11268 \@@_msg_new:nnn { Unknown-key-for~xdots }
11269 {
11270     Unknown-key.\
11271     The-key~' \l_keys_key_str '~is-unknown-for-a-command-for-drawing-dotted-rules.\
11272     \c_@@_available_keys_str
11273 }
11274 {
11275     The-available-keys-are~(in-alphabetic-order):~
11276     'color',~
11277     'horizontal(s)-labels',~
11278     'inter',~
11279     'line-style',~
11280     'nullify',~
11281     'radius',~
11282     'shorten',~
11283     'shorten-end'~and~'shorten-start'.
11284 }

11285 \@@_msg_new:nn { Unknown-key-for-rowcolors }
11286 {
11287     Unknown-key.\
11288     As-for-now,~there-is-only-two-keys-available-here:~'cols'~and~'respect-blocks'~
11289     (and-you-try-to-use~' \l_keys_key_str ').~Your-key-will-be-ignored.
11290 }

11291 \@@_msg_new:nn { Col-outside-tabular-in-trees }
11292 {
11293     Error~with~'draw-trees-in-col' \
11294     The-number-of-column~'#1'~is-outside-your-tabular~since-the-last-column~
11295     is~\int_use:N \c@jCol.~It-will-be-ignored.
11296 }

```

```

11297 \@@_msg_new:nn { Last-col-in-trees }
11298 {
11299   Error~with~'draw-trees-in-col' \\
11300   You~can't~use~'draw-trees-in-col'~with~the~column~'#1'~ because~it's~
11301   the~last~column~of~your~tabular.~It~will~be~ignored.
11302 }
11303 \@@_msg_new:nn { label~without~caption }
11304 {
11305   You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
11306   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
11307 }
11308 \@@_msg_new:nn { W~warning }
11309 {
11310   Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
11311   (row~ \int_use:N \c@iRow ).
11312 }
11313 \@@_msg_new:nn { Construct~too~large }
11314 {
11315   Construct~too~large.\\
11316   Your~command~ \token_to_str:N #1
11317   can't~be~drawn~because~your~matrix~is~too~small.~
11318   Your~command~will~be~ignored.
11319 }
11320 \@@_msg_new:nn { underscore~after~nicematrix }
11321 {
11322   Problem~with~'underscore'.\\
11323   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
11324   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
11325   ' \token_to_str:N \Cdots \token_to_str:N _
11326   \{ n \token_to_str:N \text \{ ~times \} \}' .
11327 }
11328 \@@_msg_new:nn { ampersand~in~light-syntax }
11329 {
11330   Ampersand~forbidden.\\
11331   You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
11332   ~the~key~'light-syntax'~is~in~force.
11333 }
11334 \@@_msg_new:nn { double~backslash~in~light-syntax }
11335 {
11336   Double~backslash~forbidden.\\
11337   You~can't~use~ \token_to_str:N \\
11338   ~to~separate~rows~because~the~key~'light-syntax'~
11339   is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
11340   (set~by~the~key~'end-of-row').
11341 }
11342 \@@_msg_new:nn { bad~value~for~baseline }
11343 {
11344   Bad~value~for~baseline.\\
11345   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
11346   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
11347   \int_use:N \g_@@_row_total_int \ or~equal~to~'t',~'c'~or~'b'~or~of~
11348   the~form~'line-i'.~A~value~of~1~will~be~used.
11349 }
11350 \@@_msg_new:nn { bad~value~for~baseline~line }
11351 {
11352   Bad~value~for~baseline~with~line.\\
11353   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
11354   valid.~The~number~of~the~line~must~be~between~1~and~
11355   \int_eval:n { \c@iRow + 1 }.~
11356   A~value~of~'line-1'~will~be~used.
11357 }

```

```

11358 \@@_msg_new:nn { detection-of-empty-cells }
11359 {
11360   Problem-with~'not-empty'\
11361   For-technical-reasons,~you-must-activate~
11362   'create-cell-nodes'~in~ \token_to_str:N \CodeBefore \
11363   in-order-to-use-the-key~' \l_keys_key_str '~
11364   Your-key-will-be-ignored.
11365 }
11366 \@@_msg_new:nn { siunitx-too-old }
11367 {
11368   siunitx-too-old\
11369   You-can't-use-the-columns~'S'~because-your-version-of~'siunitx'~
11370   is-too-old.~You-need-at-least-the-version-3.5.1~(2026/03/26)-and~
11371   the-version-that-you-have-loaded-has-the-date:~
11372   \str_range:cnn { ver @ siunitx . sty } { 1 } { 10 }.
11373 }
11374 \@@_msg_new:nn { Invalid-name }
11375 {
11376   Invalid-name.\
11377   You-can't-give-the-name~' \l_keys_value_tl '~to-a~ \token_to_str:N
11378   \SubMatrix \ of~your~ \@@_full_name_env: .~
11379   A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\
11380   Your-key-will-be-ignored.
11381 }
11382 \@@_msg_new:nn { Hbrace-not-allowed }
11383 {
11384   Command-not-allowed.\
11385   You-can't-use-the-command~ \token_to_str:N #1
11386   because-you-have-not-loaded~
11387   \IfPackageLoadedTF { tikz }
11388     { the-TikZ-library~'decorations.pathreplacing'~Use~ }
11389     { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
11390   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \
11391   Your-command-will-be-ignored.
11392 }
11393 \@@_msg_new:nn { Vbrace-not-allowed }
11394 {
11395   Command-not-allowed.\
11396   You-can't-use-the-command~ \token_to_str:N \Vbrace \
11397   because-you-have-not-loaded-TikZ~
11398   and-the-TikZ-library~'decorations.pathreplacing'~
11399   Use: ~\token_to_str:N \usepackage \{tikz\}~
11400   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \
11401   Your-command-will-be-ignored.
11402 }
11403 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
11404 {
11405   Wrong-line.\
11406   You-try-to-draw-a~#1~line-of-number~'#2'~in-a~
11407   \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
11408   number-is-not-valid.~It-will-be-ignored.
11409 }
11410 \@@_msg_new:nn { Impossible-SubMatrix }
11411 {
11412   Impossible-SubMatrix.\
11413   It's-impossible-to-draw-your~\token_to_str:N \SubMatrix \
11414   because-all-the-cells-are-empty-in-the-column-on-the~#1~
11415   side-of-that~\token_to_str:N \SubMatrix.
11416   \bool_if:NT \l_@@_submatrix_slim_bool
11417     { ~Maybe-you-should-try-without-the-key~'slim'. } \
11418   This~ \token_to_str:N \SubMatrix \ will-be-ignored.
11419 }

```

```

11420 \@@_msg_new:nn { Impossible-SubMatrix-no-cell-nodes }
11421 {
11422   Impossible-SubMatrix.\
11423   It's~impossible-to-draw~your~ \token_to_str:N \SubMatrix \
11424   \c_@@_explanation_no_cell_nodes_str.~
11425   This~ \token_to_str:N \SubMatrix \ will~be-ignored.
11426 }
11427 \@@_msg_new:nnn { width-without-X-columns }
11428 {
11429   You-have-used-the-key~'width'~but~you-have-put-no~'X'~column-in~
11430   the-preamble~(' \exp_not:V \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .~
11431   Your-key~will~be~ignored.
11432 }
11433 {
11434   This-message-is~the-message~'width-without-X-columns'~
11435   of~the~module~'nicematrix'.~
11436   The-experimented-users-can-disable-that-message-with~
11437   \token_to_str:N \msg_redirect_name:nnn .\
11438 }
11439
11440 \@@_msg_new:nn { key-multiplicity-with-dotted }
11441 {
11442   Incompatible-keys. \
11443   You-have-used-the-key~'multiplicity'~with~the-key~'dotted'~
11444   in~a~'custom-line'.~They~are~incompatible.~
11445   The-key~'multiplicity'~will~be~ignored.
11446 }
11447 \@@_msg_new:nn { empty-environment }
11448 {
11449   Empty-environment.\
11450   Your~ \@@_full_name_env: \ is~empty.
11451 }
11452 \@@_msg_new:nn { No-letter-and-no-command }
11453 {
11454   Erroneous-use.\
11455   Your~use-of~'custom-line'~is~no-op~since~you~don't~have~used~the~
11456   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
11457   '~ccommand'~(to~draw~horizontal~rules).~However,~you~can~go~on.
11458 }
11459 \@@_msg_new:nn { Forbidden-letter }
11460 {
11461   Forbidden-letter.\
11462   You~can't~use~the~letter~'#1'~for~a~customized~line.~
11463   It~will~be~ignored.~The~forbidden~letters~are:~\c_@@_forbidden_letters_str
11464 }
11465 \@@_msg_new:nn { Several-letters }
11466 {
11467   Wrong-name.\
11468   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
11469   have~used~' \l_@@_letter_str ').~It~will~be~ignored.
11470 }
11471 \@@_msg_new:nn { Delimiter-with-small }
11472 {
11473   Delimiter~forbidden.\
11474   You~can't~put~a~delimiter~in~the~preamble~of~your~
11475   \@@_full_name_env: \
11476   because~the~key~'small'~is~in~force.
11477 }
11478 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
11479 {

```

```

11480     Unknown~cell.\\
11481     Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
11482     the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
11483     can't~be~executed~because~a~cell~doesn't~exist.~
11484     Your~command~ \token_to_str:N \line \ will~be~ignored.
11485 }
11486 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
11487 {
11488     Duplicate~name. \\
11489     The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
11490     in~this~ \@@_full_name_env: .~Your~key~will~be~ignored.~
11491     \bool_if:NF \g_@@_messages_for_Overleaf_bool
11492     { For~a~list~of~the~names~already~used,~type~H~<return>. }
11493 }
11494 {
11495     The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
11496     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
11497 }
11498 \@@_msg_new:nn { r~or~l~with~preamble }
11499 {
11500     Erroneous~use. \\
11501     You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
11502     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
11503     your~ \@@_full_name_env: .~
11504     Your~key~will~be~ignored.
11505 }
11506 \@@_msg_new:nn { Hdotsfor~in~col~0 }
11507 {
11508     Erroneous~use. \\
11509     You~can't~use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\
11510     in~an~exterior~column~of~the~array.
11511 }
11512 \@@_msg_new:nn { bad~corner }
11513 {
11514     Bad~corner. \\
11515     '#1'~is~an~incorrect~specification~for~a~corner~(in~the~key~
11516     'corners').~The~available~values~are:~NW,~SW,~NE,~SE,~N,~S,~W~and~E~
11517     This~specification~of~corner~will~be~ignored.
11518 }
11519 \@@_msg_new:nn { bad~border }
11520 {
11521     Bad~border.\\
11522     '\l_keys_key_str'~is~an~incorrect~specification~for~a~border~
11523     (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
11524     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
11525     also~use~the~key~'tikz'
11526     \IfPackageLoadedF { tikz }
11527     { ~if~you~load~the~LaTeX~package~'tikz' } ).~
11528     This~specification~of~border~will~be~ignored.
11529 }
11530 \@@_msg_new:nn { TikzEveryCell~without~tikz }
11531 {
11532     TikZ~not~loaded. \\
11533     You~can't~use~ \token_to_str:N \TikzEveryCell \
11534     because~you~have~not~loaded~TikZ.~You~can~load~TikZ~with~
11535     \token_to_str:N \usepackage \{tikz\},~before~or~after~'nicematrix'.~
11536     Your~command~will~be~ignored.
11537 }
11538 \@@_msg_new:nn { tikz~key~without~tikz }
11539 {
11540     TikZ~not~loaded. \\

```

```

11541     You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N
11542     \Block '~because~you~have~not~loaded~TikZ.~
11543     You~can~load~TikZ~with~\token_to_str:N \usepackage \{tikz\},~
11544     before~or~after~'nicematrix'.~Your~key~will~be~ignored.
11545   }
11546   \@@_msg_new:nn { Bad~argument~for~Block }
11547   {
11548     Bad~argument. \\
11549     The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
11550     be~of~the~form~'i-j'~(or~totally~empty)~and~you~have~used:~
11551     '#1'.~ If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono~cell~(as~if~
11552     the~argument~was~empty).
11553   }
11554   \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
11555   {
11556     Erroneous~use. \\
11557     In~the~ \@@_full_name_env: ,~you~must~use~the~key~
11558     'last~col'~without~value.~However,~you~can~go~on~for~this~time~
11559     (the~value~' \l_keys_value_tl '~will~be~ignored).
11560   }
11561   \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
11562   {
11563     Erroneous~use. \\
11564     In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
11565     'last~col'~without~value.~ However,~you~can~go~on~for~this~time~
11566     (the~value~' \l_keys_value_tl '~will~be~ignored).
11567   }
11568   \@@_msg_new:nn { Block~too~large~1 }
11569   {
11570     Block~too~large. \\
11571     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
11572     too~small~for~that~block.~This~block~and~maybe~others~will~be~ignored.
11573   }
11574   \@@_msg_new:nn { Block~too~large~2 }
11575   {
11576     Block~too~large. \\
11577     The~preamble~of~your~ \@@_full_name_env:,~
11578     which~is~ '\exp_not:V\g_@@_user_preamble_tl',~announces~
11579     \int_use:N \g_@@_static_num_of_col_int \ columns~
11580     \@@_message_about_false_cols:
11581     but~you~use~only~ \int_use:N \c@jCol \ and~that's~why~a~block~
11582     specified~in~the~cell~#1-#2~can't~be~drawn.~
11583     \bool_if:NF \l_@@_light_syntax_bool
11584     {
11585       You~should~add~some~ampersands~(&)~at~the~end~of~the~first~row~
11586       of~your~\@@_full_name_env:~
11587     }
11588     This~block~will~be~ignored.
11589   }
11590   \@@_msg_new:nn { unknown~column~type }
11591   {
11592     Bad~column~type. \\
11593     The~column~type~'#1'~in~your~ \@@_full_name_env: \ is~unknown.
11594   }
11595   \@@_msg_new:nn { unknown~column~type~multicolumn }
11596   {
11597     Bad~column~type. \\
11598     The~column~type~'#1'~in~the~command~\token_to_str:N \multicolumn \
11599     ~of~your~ \@@_full_name_env: \ is~unknown.
11600   }

```

```

11601 \@@_msg_new:nn { unknown~column~type~S }
11602 {
11603   Bad~column~type. \\
11604   The~column~type~'S'~in~your~ \@@_full_name_env: \ is~unknown.~
11605   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
11606   load~that~package~with~\token_to_str:N \usepackage \{siunitx\}.
11607 }
11608 \@@_msg_new:nn { unknown~column~type~S~multicolumn }
11609 {
11610   Bad~column~type. \\
11611   The~column~type~'S'~in~the~command~\token_to_str:N \multicolumn \
11612   of~your~ \@@_full_name_env: \ is~unknown.~If~you~want~to~use~the~column~
11613   type~'S'~of~siunitx,~you~should~load~that~package~(with~
11614   \token_to_str:N \usepackage \{siunitx\}).
11615 }
11616 \@@_msg_new:nn { unknown~column~type~; }
11617 {
11618   Bad~column~type. \\
11619   The~column~type~';'~in~your~ \@@_full_name_env: \ is~unknown.~
11620   You~should~load~TikZ~(with~\token_to_str:N \usepackage \{tikz\})~
11621   in~order~to~use~it~for~vertical~dashed~rules.
11622 }
11623 \@@_msg_new:nn { unknown~column~type~RCL }
11624 {
11625   Bad~column~type. \\
11626   The~column~type~'#1'~in~your~ \@@_full_name_env: \ is~unknown.~
11627   Maybe~you~want~to~use~\str_lowercase:n { #1 }.
11628 }
11629 \@@_msg_new:nn { tabularnote~forbidden }
11630 {
11631   Forbidden~command. \\
11632   You~can't~use~the~command~ \token_to_str:N \tabularnote \
11633   ~here.~This~command~is~available~only~in~
11634   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
11635   the~argument~of~a~command~\token_to_str:N \caption \ included~
11636   in~an~environment~\{table\}.~Your~command~will~be~ignored.
11637 }
11638 \@@_msg_new:nn { borders~forbidden }
11639 {
11640   Forbidden~key.\\
11641   You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \
11642   because~the~option~'rounded~corners'~is~in~force~with~a~non~zero~value.~
11643   Your~key~will~be~ignored.
11644 }
11645 \@@_msg_new:nn { bottomrule~without~booktabs }
11646 {
11647   booktabs~not~loaded.\\
11648   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
11649   loaded~'booktabs'.~You~should~load~'booktabs',~before~or~
11650   after~'nicematrix'.~Your~key~will~be~ignored.
11651 }
11652 \@@_msg_new:nn { enumitem~not~loaded }
11653 {
11654   enumitem~not~loaded. \\
11655   You~can't~use~the~command~ \token_to_str:N \tabularnote \
11656   ~because~you~haven't~loaded~'enumitem'.~We~should~load~it~
11657   (before~or~after~'nicematrix').~All~the~commands~
11658   \token_to_str:N \tabularnote \ will~be~ignored~in~the~document.
11659 }
11660 \@@_msg_new:nn { tikz~without~tikz }

```

```

11661 {
11662   TikZ~not~loaded. \\
11663   You~can~t~use~the~key~'tikz'~here~because~TikZ~is~not~loaded.~You~
11664   should~load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.~
11665   If~you~go~on,~your~key~will~be~ignored.
11666 }
11667 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
11668 {
11669   TikZ~not~loaded. \\
11670   You~have~used~the~key~'tikz'~(or~'dashed')~in~the~definition~of~a~
11671   customized~line~(with~'custom~line')~but~TikZ~is~not~loaded.~
11672   You~can~go~on~but~you~will~have~another~error~if~you~actually~
11673   use~that~custom~line.~You~should~load~TikZ~(with~
11674   \token_to_str:N \usepackage \{tikz\}).
11675 }
11676 \@@_msg_new:nn { tikz~in~borders~without~tikz }
11677 {
11678   TikZ~not~loaded. \\
11679   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
11680   command~' \token_to_str:N \Block ')~but~TikZ~is~not~loaded.~
11681   Your~key~will~be~ignored.~You~should~load~TikZ~(with~
11682   \token_to_str:N \usepackage \{tikz\}).
11683 }
11684 \@@_msg_new:nn { color~in~custom~line~with~tikz }
11685 {
11686   Erroneous~use.\\
11687   In~a~'custom~line',~you~have~used~both~'tikz'~(or~'dashed')~and~'color',~
11688   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz'~itself).~
11689   The~key~'color'~will~be~ignored.
11690 }
11691 \@@_msg_new:nn { Wrong~last~row }
11692 {
11693   Wrong~number.\\
11694   You~have~used~'last~row= \int_use:N \l_@@_last_row_int '~but~your~
11695   \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
11696   If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
11697   last~row~but~you~should~correct~your~code.~You~can~avoid~this~
11698   problem~by~using~'last~row'~without~value~(more~compilations~
11699   might~be~necessary).
11700 }
11701 \@@_msg_new:nn { Yet~in~env }
11702 {
11703   Nested~environments.\\
11704   Environments~of~nicematrix~can't~be~nested.~However~you~
11705   can~insert,~for~example,~a~\{tabular\}~in~a~\{NiceTabular\}~
11706   or~a~\{NiceTabular\}~in~a~\{tabular\}.~You~can~also~compose~
11707   an~environment~of~nicematrix~in~a~box~of~LaTeX~and~insert~
11708   that~box~in~another~environment~of~nicematrix.
11709 }
11710 \@@_msg_new:nn { Outside~math~mode }
11711 {
11712   Outside~math~mode.\\
11713   The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
11714   (and~not~in~ \token_to_str:N \vcenter ).
11715 }
11716 \@@_msg_new:nn { One~letter~allowed }
11717 {
11718   Bad~name.\\
11719   The~value~of~key~' \l_keys_key_str '~must~be~of~length~1~and~
11720   you~have~used~' \l_keys_value_tl '~.~Your~key~will~be~ignored.
11721 }

```

```

11722 \@@_msg_new:nn { TabularNote~in~CodeAfter }
11723 {
11724   Environment~\{TabularNote\}~forbidden.\\
11725   You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
11726   but~*before*~the~ \token_to_str:N \CodeAfter .~Your~environment~
11727   \{TabularNote\}~will~be~ignored.
11728 }
11729 \@@_msg_new:nn { varwidth~not~loaded }
11730 {
11731   varwidth~not~loaded.\\
11732   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
11733   loaded.~You~should~load~'varwidth',~before~or~after~'nicematrix'.~
11734   Your~column~will~behave~like~'p'.
11735 }
11736 \@@_msg_new:nn { varwidth~not~loaded~in~X }
11737 {
11738   varwidth~not~loaded.\\
11739   You~can't~use~the~key~'V'~in~your~column~'X'~
11740   because~'varwidth'~is~not~loaded.~You~should~load~'varwidth',~
11741   before~or~after~'nicematrix'.~ Your~key~will~be~ignored.
11742 }
11743 \@@_msg_new:nnn { Unknown~key~for~a~rule }
11744 {
11745   Unknown~key.\\
11746   Your~key~' \l_keys_key_str 'is~unknown~for~a~rule.\\
11747   \c_@@_available_keys_str
11748 }
11749 {
11750   The~available~keys~are:~color,~multiplicity,~sep~color,~tikz~
11751   and~total~width~(the~latter~is~meaningful~only~in~conjunction~with~'tikz').
11752 }

```

If fact, there is also the key dotted but it won't be very useful since we provide `\hdottedline`, `\cdottedline` and the letter `:`.

```

11753 \@@_msg_new:nnn { Unknown~key~for~Block }
11754 {
11755   Unknown~key. \\
11756   The~key~' \l_keys_key_str 'is~unknown~for~the~command~
11757   \token_to_str:N \Block .~Your~key~will~be~ignored. \\
11758   \c_@@_available_keys_str
11759 }
11760 {
11761   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
11762   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~name,~
11763   opacity,~rounded~corners,~r,~respect~arraystretch,~rules/color,~rules/width,~
11764   t,~T,~tikz,~transparent~and~vlines.
11765 }
11766 \@@_msg_new:nnn { Unknown~key~for~Brace }
11767 {
11768   Unknown~key.\\
11769   The~key~' \l_keys_key_str 'is~unknown~for~the~commands~
11770   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace .~
11771   Your~key~will~be~ignored. \\
11772   \c_@@_available_keys_str
11773 }
11774 {
11775   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
11776   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
11777   right~shorten)~and~yshift.
11778 }

```

```

11779 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
11780 {
11781   Unknown~key.\\
11782   The~key~' \l_keys_key_str '~is~unknown.~It~will~be~ignored. \\
11783   \c_@@_available_keys_str
11784 }
11785 {
11786   The~available~keys~are~(in~alphabetic~order):~
11787   delimiters/color,~
11788   rules~(with~the~subkeys~'color'~and~'width'),~
11789   sub-matrix~(several~subkeys)~
11790   and~xdots~(several~subkeys).~
11791   The~latter~is~for~the~command~ \token_to_str:N \line .
11792 }
11793 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
11794 {
11795   Unknown~key.\\
11796   The~key~' \l_keys_key_str '~is~unknown.~Your~key~will~be~ignored. \\
11797   \c_@@_available_keys_str
11798 }
11799 {
11800   The~available~keys~are~(in~alphabetic~order):~
11801   create-cell-nodes,~
11802   delimiters/color~and~
11803   sub-matrix~(several~subkeys).
11804 }
11805 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
11806 {
11807   Unknown~key.\\
11808   The~key~' \l_keys_key_str '~is~unknown.~Your~key~will~be~ignored. \\
11809   \c_@@_available_keys_str
11810 }
11811 {
11812   The~available~keys~are~(in~alphabetic~order):~
11813   'delimiters/color',~
11814   'extra-height',~
11815   'hlines',~
11816   'hvlines',~
11817   'left-xshift',~
11818   'name',~
11819   'right-xshift',~
11820   'rules'~(with~the~subkeys~'color'~and~'width'),~
11821   'slim',~
11822   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~and~'right-xshift').
11823 }
11824 \@@_msg_new:nnn { Unknown~key~for~notes }
11825 {
11826   Unknown~key.\\
11827   The~key~' \l_keys_key_str '~is~unknown.~ Your~key~will~be~ignored. \\
11828   \c_@@_available_keys_str
11829 }
11830 {
11831   The~available~keys~are~(in~alphabetic~order):~
11832   bottomrule,~
11833   code-after,~
11834   code-before(+),~
11835   detect-duplicates,~
11836   enumitem-keys,~
11837   enumitem-keys-para,~
11838   para,~
11839   label-in-list,~
11840   label-in-tabular~and~
11841   style.

```

```

11842 }
11843 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
11844 {
11845   Unknown~key.\\
11846   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11847   \token_to_str:N \RowStyle .~Your~key~will~be~ignored. \\
11848   \c_@@_available_keys_str
11849 }
11850 {
11851   The~available~keys~are~(in~alphabetic~order):~
11852   bold,~
11853   cell-space-top-limit(+),~
11854   cell-space-bottom-limit(+),~
11855   cell-space-limits(+),~
11856   color,~
11857   fill~(alias:~rowcolor),~
11858   nb-rows,~
11859   opacity~and~
11860   rounded-corners.
11861 }
11862 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
11863 {
11864   Unknown~key.\\
11865   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11866   \token_to_str:N \NiceMatrixOptions .~Your~key~will~be~ignored. \\
11867   \c_@@_available_keys_str
11868 }
11869 {
11870   The~available~keys~are~(in~alphabetic~order):~
11871   &~in~blocks,~
11872   allow-duplicate-names,~
11873   ampersand-in-blocks,~
11874   caption-above,~
11875   cell-space-bottom-limit(+),~
11876   cell-space-limits(+),~
11877   cell-space-top-limit(+),~
11878   code-for-first-col(+),~
11879   code-for-first-row(+),~
11880   code-for-last-col(+),~
11881   code-for-last-row(+),~
11882   corners,~
11883   custom-key,~
11884   create-extra-nodes,~
11885   create-medium-nodes,~
11886   create-large-nodes,~
11887   custom-line,~
11888   delimiters~(several~subkeys),~
11889   end-of-row,~
11890   first-col,~
11891   first-row,~
11892   h(v)lines,~
11893   h(v)lines-except-borders,~
11894   last-col,~
11895   last-row,~
11896   left-margin,~
11897   light-syntax,~
11898   light-syntax-expanded,~
11899   matrix/columns-type,~
11900   no-cell-nodes,~
11901   notes~(several~subkeys),~
11902   nullify-dots,~
11903   pgf-node-code,~
11904   renew-dots,~

```

```

11905     renew-matrix,~
11906     respect-arraystretch,~
11907     rounded-corners,~
11908     right-margin,~
11909     rules~(with~the~subkeys~'color'~and~'width'),~
11910     small,~
11911     sub-matrix~(several~subkeys),~
11912     vlines,~
11913     width-of-false(+),~
11914     xdots~(several~subkeys).
11915 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

11916 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
11917 {
11918   Unknown~key.\\
11919   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11920   \{NiceArray\}.~Your~key~will~be~ignored. \\
11921   \c_@@_available_keys_str
11922 }
11923 {
11924   The~available~keys~are~(in~alphabetic~order):~
11925   &~in~blocks,~
11926   ampersand~in~blocks,~
11927   b,~
11928   baseline,~
11929   c,~
11930   cell-space-bottom-limit,~
11931   cell-space-limits,~
11932   cell-space-top-limit,~
11933   code-after,~
11934   code-for-first-col(+),~
11935   code-for-first-row(+),~
11936   code-for-last-col(+),~
11937   code-for-last-row(+),~
11938   columns-width,~
11939   corners,~
11940   create-blocks-in-col,~
11941   create-extra-nodes,~
11942   create-medium-nodes,~
11943   create-large-nodes,~
11944   draw-trees-in-col,~
11945   extra-left-margin,~
11946   extra-right-margin,~
11947   first-col,~
11948   first-row,~
11949   h(v)lines,~
11950   h(v)lines-except-borders,~
11951   last-col,~
11952   last-row,~
11953   left-margin,~
11954   light-syntax,~
11955   light-syntax-expanded,~
11956   name,~
11957   no-cell-nodes,~
11958   nullify-dots,~
11959   pgf-node-code,~
11960   renew-dots,~
11961   respect-arraystretch,~
11962   right-margin,~
11963   rounded-corners,~
11964   rules~(with~the~subkeys~'color'~and~'width'),~
11965   small,~

```

```

11966     t,~
11967     vlines,~
11968     width-of-false(+),~
11969     xdots/color,~
11970     xdots/shorten-start(+),~
11971     xdots/shorten-end(+),~
11972     xdots/shorten(+)-and~
11973     xdots/line-style.
11974 }

```

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`).

```

11975 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
11976 {
11977   Unknown~key.\\
11978   The~key~' \l_keys_key_str '-is~unknown~for~the~
11979   \@@_full_name_env: .~Your~key~will~be~ignored. \\
11980   \c_@@_available_keys_str
11981 }
11982 {
11983   The~available~keys~are~(in~alphabetic~order):~
11984   &~in~blocks,~
11985   ampersand~in~blocks,~
11986   b,~
11987   baseline,~
11988   c,~
11989   cell-space-bottom-limit,~
11990   cell-space-limits,~
11991   cell-space-top-limit,~
11992   code-after,~
11993   code-for-first-col(+),~
11994   code-for-first-row(+),~
11995   code-for-last-col(+),~
11996   code-for-last-row(+),~
11997   columns-type,~
11998   columns-width,~
11999   corners,~
12000   create-blocks-in-col,~
12001   create-extra-nodes,~
12002   create-medium-nodes,~
12003   create-large-nodes,~
12004   draw-trees-in-col,~
12005   extra-left-margin,~
12006   extra-right-margin,~
12007   first-col,~
12008   first-row,~
12009   h(v)lines,~
12010   h(v)lines-except-borders,~
12011   l,~
12012   last-col,~
12013   last-row,~
12014   left-margin,~
12015   light-syntax,~
12016   light-syntax-expanded,~
12017   name,~
12018   no-cell-nodes,~
12019   nullify-dots,~
12020   pgf-node-code,~
12021   r,~
12022   renew-dots,~
12023   respect-arraystretch,~
12024   right-margin,~
12025   rounded-corners,~

```

```

12026 rules~(with~the~subkeys~'color'~and~'width'),~
12027 small,~
12028 t,~
12029 vlines,~
12030 width-of-false(+),~
12031 xdots/color,~
12032 xdots/shorten-start(+),~
12033 xdots/shorten-end(+),~
12034 xdots/shorten(+),~and~
12035 xdots/line-style.
12036 }
12037 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
12038 {
12039   Unknown~key.\\
12040   The~key~' \l_keys_key_str 'is~unknown~for~the~environment~
12041   \{NiceTabular\}.~Your~key~will~be~ignored. \\
12042   \c_@@_available_keys_str
12043 }
12044 {
12045   The~available~keys~are~(in~alphabetic~order):~
12046   &-in-blocks,~
12047   ampersand-in-blocks,~
12048   b,~
12049   baseline,~
12050   c,~
12051   caption,~
12052   cell-space-bottom-limit,~
12053   cell-space-limits,~
12054   cell-space-top-limit,~
12055   code-after,~
12056   code-for-first-col(+),~
12057   code-for-first-row(+),~
12058   code-for-last-col(+),~
12059   code-for-last-row(+),~
12060   columns-width,~
12061   corners,~
12062   custom-line,~
12063   create-blocks-in-col,~
12064   create-extra-nodes,~
12065   create-medium-nodes,~
12066   create-large-nodes,~
12067   draw-trees-in-col,~
12068   extra-left-margin,~
12069   extra-right-margin,~
12070   first-col,~
12071   first-row,~
12072   h(v)lines,~
12073   h(v)lines-except-borders,~
12074   label,~
12075   last-col,~
12076   last-row,~
12077   left-margin,~
12078   light-syntax,~
12079   light-syntax-expanded,~
12080   name,~
12081   no-cell-nodes,~
12082   notes~(several~subkeys),~
12083   nullify-dots,~
12084   pgf-node-code,~
12085   renew-dots,~
12086   respect-arraystretch,~
12087   right-margin,~
12088   rounded-corners,~

```

```

12089 rules~(with~the~subkeys~'color'~and~'width'),~
12090 short-caption,~
12091 t,~
12092 tabularnote,~
12093 vlines,~
12094 width-of-false(+),~
12095 xdots/color,~
12096 xdots/shorten-start(+),~
12097 xdots/shorten-end(+),~
12098 xdots/shorten(+)-and~
12099 xdots/line-style.
12100 }
12101 \@@_msg_new:nnn { Duplicate-name }
12102 {
12103 Duplicate-name.\\
12104 The-name~' \l_keys_value_tl '~is-already-used-and-you-shouldn't-use~
12105 the-same-environment-name-twice.~You-can-go-on,~but,~
12106 maybe,~you-will-have-incorrect-results-especially~
12107 if-you-use~'columns-width=auto'.~If-you-don't-want-to-see-this~
12108 message-again,~use~the-key~'allow-duplicate-names'~in~
12109 ' \token_to_str:N \NiceMatrixOptions '.\\
12110 \bool_if:NF \g_@@_messages_for_Overleaf_bool
12111 { For-a-list-of-the-names-already-used,~type-H~<return>. }
12112 }
12113 {
12114 The-names-already-defined-in-this-document-are:~
12115 \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
12116 }
12117 \@@_msg_new:nn { caption-above-in-env }
12118 {
12119 The-key~'caption-above'~must-be-used-in~\token_to_str:N \NiceMatrixOptions.~
12120 Your-key-will-be-ignored.
12121 }
12122 \@@_msg_new:nn { show-cell-names }
12123 {
12124 There-is-no-key~'show-cell-names'~in-nicematrix.\\
12125 You-should-use-the-command~\token_to_str:N \ShowCellNames~
12126 in-the~\token_to_str:N \CodeBefore~ or~the~\token_to_str:N
12127 \CodeAfter.~Your-key-will-be-ignored.
12128 }
12129 \@@_msg_new:nn { Option-auto-for-columns-width }
12130 {
12131 Erroneous-use.\\
12132 You-can't-give-the-value~'auto'~to-the-key~'columns-width'~here.~
12133 Your-key-will-be-ignored.
12134 }
12135 \@@_msg_new:nn { NiceTabularX~without~X }
12136 {
12137 NiceTabularX-without-X.\\
12138 You-should-not-use~\{NiceTabularX\}~without-X-columns.~However,~you-can-go-on.
12139 }
12140 \@@_msg_new:nn { Preamble-forgotten }
12141 {
12142 Preamble-forgotten.\\
12143 You-have-probably-forgotten-the-preamble-of-your~
12144 \@@_full_name_env: .
12145 }
12146 \@@_msg_new:nn { Invalid-col-number }
12147 {
12148 Invalid-column-number.\\
12149 A-color-instruction-in-the~ \token_to_str:N \CodeBefore \

```

```

12150     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.~
12151     Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
12152 }
12153 \@@_msg_new:nn { Invalid~row~number }
12154 {
12155     Invalid~row~number.\\
12156     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
12157     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.~
12158     Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
12159 }
12160 \@@_define_com:NNN p ( )
12161 \@@_define_com:NNN b [ ]
12162 \@@_define_com:NNN v | |
12163 \@@_define_com:NNN V \l \l
12164 \@@_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>	82
13	The environment <code>{NiceMatrix}</code> and its variants	100
	13.1 Definition of <code>{pNiceMatrix}</code>	100
	13.2 The key <code>renew-matrix</code>	101
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	101
15	After the construction of the array	102
16	We draw the dotted lines	112
17	The actual instructions for drawing the dotted lines with <code>TikZ</code>	129
18	User commands available in the new environments	135
19	The command <code>\line</code> accessible in <code>\CodeAfter</code>	141
20	The command <code>\RowStyle</code>	143
21	Colors of cells, rows and columns	146
22	The vertical and horizontal rules	158
23	The empty corners	181
24	The environment <code>{NiceMatrixBlock}</code>	185
25	The extra nodes	186
26	The blocks	191
27	Automatic arrays	219
28	The redefinition of the command <code>\dotfill</code>	220
29	The command <code>\diagbox</code>	221

30	The keyword <code>\CodeAfter</code>	222
31	The delimiters in the preamble	223
32	The command <code>\SubMatrix</code>	224
33	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	233
34	The commands <code>HBrace</code> et <code>VBrace</code>	236
35	The command <code>TikzEveryCell</code>	239
36	The key <code>draw-trees-in-col</code>	241
37	The key <code>create-blocks-in-col</code>	242
38	The command <code>\ShowCellNames</code>	243
39	We process the options at package loading	245
40	About the package underscore	247
41	Compatibility with <code>threeparttable</code>	247
42	Gestion of the errors	247
	42.1 Security in the <code>CodeBefore</code> and the <code>CodeAfter</code>	247
	42.2 The error messages of the package	248