

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

April 23, 2026

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

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

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

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

*This document corresponds to the version 7.8b of `nicematrix`, at the date of 2026/04/23.

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

```

20 \RequirePackage { amsmath }

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

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

```

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

```

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

```

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

```

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

```

2 Collecting options

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

Example :

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

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

```

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

```

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

```

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

```

3 Technical definitions

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

```

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

103 \tl_const:Nn \c_@@_brace_tl { nicematrix/brace }
104 \tl_const:Nn \c_@@_mirrored_brace_tl { nicematrix/mirrored-brace }

```

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

```

105 \tl_new:N \l_@@_argspec_tl

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

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

```

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

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

172 \AtBeginDocument

```

```

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

```

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

```

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

```

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

```

196     }
197   }
198   {
199     \cs_new_protected:Npn \@@_revert_colortbl: { }
200     \cs_new_protected:Npn \@@_replace_columncolor:
201     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }

```

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

```

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

```

Idem for `\CT@drs@`.

```

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

```

```

225     }
226 }

```

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

```

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

```

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

```

237     \skip_horizontal:N \c_zero_dim
238 }

```

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

```

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

```

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

```

243 \cs_set:Npn \@@_cline:

```

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

```

244 { \@@_cline_i:en { \l_@@_first_col_int } }

```

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

```

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

```

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

```

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

```

¹See question 99041 on TeX StackExchange.

```

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

```

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

```

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

```

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

```

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

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```


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

```

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

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

310 \int_new:N \l_@@_tmpc_int

```

4 Parameters

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

```

311 \int_new:N \g_@@_env_int

```

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

```

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

```

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

```

313 \NewExpandableDocumentCommand \NiceMatrixLastEnv { } { \int_use:N \g_@@_env_int }

```

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

```

314 \box_new:N \l_@@_the_array_box

```

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

```

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

```

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

```

317 \bool_new:N \l_@@_tabular_bool

```

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

```

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

```

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

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

```

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

```

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

```

322 \bool_new:N \l_@@_NiceMatrix_without_vlines_bool

```

The following counter will count the environments {NiceMatrixBlock}.

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

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

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

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

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

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

```
326 \dim_new:N \l_@@_col_width_dim
327 \dim_set:Nn \l_@@_col_width_dim { -1 cm }
```

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

```
328 \int_new:N \g_@@_row_total_int
329 \int_new:N \g_@@_col_total_int
```

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

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

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

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

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

```
332 \tl_new:N \l_@@_hpos_cell_tl
333 \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
```

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

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

Idem for the mono-row blocks.

```
335 \dim_new:N \g_@@_blocks_ht_dim
336 \dim_new:N \g_@@_blocks_dp_dim
```

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

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

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

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

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

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

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

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

```
342 \bool_new:N \l_@@_initial_open_bool
343 \bool_new:N \l_@@_final_open_bool
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
350 \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.

```
351 \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`).

```
352 \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`.

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

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

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

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

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

```
356 \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).

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

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

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

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

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

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

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

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

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

For `\multicolumn`.

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

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

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

```
365 \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 `:`.

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

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

```
368 \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`).

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

```
370 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
371 {
```

```
372   \if_mode_math: \else:
```

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

```
374   \fi:
```

```
375 }
```

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.

```
376 \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”.

```
377 \colorlet { nicematrix-last-col } { . }
378 \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`).

```
379 \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*.

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

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

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

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

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

```
384 \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)`.

```
385 \cs_new:Npn \@@_full_name_env:
386 {
387   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
388     { command \space \c_backslash_str \g_@@_name_env_str }
389     { environment \space \{ \g_@@_name_env_str \} }
390 }
```

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

The argument is given by curryfication.

```
392 \cs_new_protected:Npn \@@_put_in_cell_after_hook:n
393 { \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.

```
394 \tl_new:N \g_@@_pre_code_before_tl
395 \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`.

```
396 \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.

```
397 \tl_new:N \g_@@_pre_code_after_tl
398 \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.

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

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

```
400 \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.

```
401 \int_new:N \l_@@_old_iRow_int
402 \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).

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

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

```
404 \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.

```
405 \bool_new:N \l_@@_X_columns_aux_bool
406 \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.

```
407 \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).

```
408 \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).

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

```
410 \tl_new:N \l_@@_code_before_tl
411 \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).

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

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

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

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

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

```
423 \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”.

```
424 \dim_new:N \g_@@_width_last_col_dim
425 \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.

```
426 \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!).

```
427 \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.

```
428 \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}`.

```
429 \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.

```
430 \seq_new:N \g_@@_pos_of_stroken_blocks_seq
```

If the user has used the key `corners`, all the cells which are in an (empty) corner will be stored in the following list. We use a `clist` instead of a `seq` because we will frequently search in that list (and searching in a `clist` is faster than searching in a `seq`).

```
431 \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`).

```
432 \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.

```
433 \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).

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

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

```
436 \int_new:N \l_@@_multiplicity_int
437 \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.

```
438 \int_new:N \g_@@_ddots_int
439 \int_new:N \g_@@_iddots_int
```

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

```
440 \dim_new:N \g_@@_delta_x_one_dim
441 \dim_new:N \g_@@_delta_y_one_dim
442 \dim_new:N \g_@@_delta_x_two_dim
443 \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).

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

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

The following counters will be used when drawing the rules.

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

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

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

```
454 \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).

```
455 \int_new:N \g_@@_static_num_of_col_int
```

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

```
456 \tl_new:N \l_@@_draw_tl
457 \seq_new:N \l_@@_tikz_seq
458 \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`).

```
459 \str_new:N \l_@@_hpos_block_str
460 \str_set:Nn \l_@@_hpos_block_str { c }
461 \bool_new:N \l_@@_hpos_of_block_cap_bool
462 \bool_new:N \l_@@_p_block_bool
```

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

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

```
464 \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).

```
465 \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.

```
466 \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}`).

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

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

```
468 \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`).

```
469 \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.

```
470 \int_new:N \l_@@_first_row_int
471 \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.

```
472 \int_new:N \l_@@_first_col_int
473 \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).

```
474 \int_new:N \l_@@_last_row_int
475 \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

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

Idem for `\l_@@_last_col_without_value_bool`

```
477 \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.

```
478 \int_new:N \l_@@_last_col_int
479 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

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

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

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

```
480 \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`).

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

Some utilities

```
482 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
483 {
```

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

```
484 \def \l_tmpa_tl { #1 }
485 \def \l_tmpb_tl { #2 }
486 }
```

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`.

```
487 \cs_new_protected:Npn \@@_expand_clist_hvlines:NN #1 #2
488 {
489   \clist_if_in:NnF #1 { all }
490   {
491     \clist_clear:N \l_tmpa_clist
492     \clist_map_inline:Nn #1
493     {
494       \tl_if_head_eq_meaning:nNTF { ##1 } -
495       {
```

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.

```
496       \int_if_zero:nF { #2 }
497       {
498         \clist_put_right:Ne \l_tmpa_clist
499         { \int_eval:n { #2 + (##1) + 1 } }
500       }
501     }
502   }
```

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

```
503       \tl_if_in:nnTF { ##1 } { - }
504       { \@@_cut_on_hyphen:w ##1 \q_stop }
505       {
```

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

```
506       \def \l_tmpa_tl { ##1 }
507       \def \l_tmpb_tl { ##1 }
508     }
509     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
510     { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
511   }
512 }
513 \tl_set_eq:NN #1 \l_tmpa_clist
514 }
515 }
```

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.

```

516 \AtBeginDocument
517 {
518     \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
519     \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
520 }

```

5 The command `\tabularnote`

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

- The caption may of course be provided by the command `\caption` in a floating environment. Of course, a command `\tabularnote` in that `\caption` makes sens only if the `\caption` is *before* the `{tabular}`.
- It’s also possible to use `\tabularnote` in the value of the key `caption` of the `{NiceTabular}` when the key `caption-above` is in force. However, in that case, one must remind that the caption is composed *after* the composition of the box which contains the main tabular (that’s mandatory since that caption must be wrapped with a line width equal to the width of the tabular). However, we want the labels of the successive tabular notes in the logical order. That’s why:

- The number of tabular notes present in the caption will be written on the `aux` file and available in `\g_@@_notes_caption_int`.⁴
- During the composition of the main tabular, the tabular notes will be numbered from `\g_@@_notes_caption_int+1` and the notes will be stored in `\g_@@_notes_seq`. Each component of `\g_@@_notes_seq` will be a kind of couple of the form : `{label}{text of the tabularnote}`. The first component is the optional argument (between square brackets) of the command `\tabularnote` (if the optional argument is not used, the value will be the special marker expressed by `\NoValue`).
- During the composition of the caption (value of `\l_@@_caption_tl`), the tabular notes will be numbered from 1 to `\g_@@_notes_caption_int` and the notes themselves will be stored in `\g_@@_notes_in_caption_seq`. The structure of the components of that sequence will be the same as for `\g_@@_notes_seq`.
- After the composition of the main tabular and after the composition of the caption, the sequences `\g_@@_notes_in_caption_seq` and `\g_@@_notes_seq` will be merged (in that order) and the notes will be composed.

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

```

521 \newcounter { tabularnote }

```

We want to avoid error messages for duplicate labels when the package `hyperref` is used. That’s why we will count all the tabular notes of the whole document with `\g_@@_tabularnote_int`.

```

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

```

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

```

524 \seq_new:N \g_@@_notes_seq
525 \seq_new:N \g_@@_notes_in_caption_seq

```

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

```

526 \tl_new:N \g_@@_tabularnote_tl

```

We prepare the tools for the formatting of the references of the footnotes (in the tabular itself). There may have several references of footnote at the same point and we have to take into account that point.

```

527 \seq_new:N \l_@@_notes_labels_seq
528 \newcounter { nicematrix_draft }
529 \cs_new_protected:Npn \@@_notes_format:n #1
530 {
531   \setcounter { nicematrix_draft } { #1 }
532   \@@_notes_style:n { nicematrix_draft }
533 }

```

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

```

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

```

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

```

535 \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`.

```

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

```

We define `\thetabularnote` because it will be used by LaTeX if the user want to reference a tabular which has been marked by a `\label`. The TeX group is for the case where the user has put an instruction such as `\color{red}` in `\@@_notes_style:n`.

```

537 \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`).

```

538 \AtBeginDocument
539 {
540   \IfPackageLoadedTF { enumitem }
541   {

```

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.

```

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

```

```

557         itemjoin = \quad ,
558         label =
559             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
560     }

```

One must remind that we have allowed a `\tabular` in the caption and that caption may also be found in the list of tables (`\listoftables`). We want the command `\tabularnote` be no-op during the composition of that list. That's why we program `\tabularnote` to be no-op excepted in a floating environment or in an environment of `nicematrix`.

```

561     \NewDocumentCommand { \tabularnote } { o m }
562     {
563         \bool_lazy_or:nnTF \l_@@_in_env_bool { \cs_if_exist_p:N \@capytype }
564         {
565             \bool_lazy_and:nnTF \l_@@_in_env_bool { ! \l_@@_tabular_bool }
566             { \@@_error:n { tabularnote~forbidden } }
567         }

```

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

```

568         \bool_if:NTF \l_@@_in_caption_bool
569         {
570             \tl_if_novalue:nTF { #1 }
571             { \@@_tabularnote_caption:nn { #1 } }
572             { \@@_tabularnote_caption:nn { \exp_not:n { #1 } } }
573         }
574         {
575             \tl_if_novalue:nTF { #1 }
576             { \@@_tabularnote:nn { #1 } }
577             { \@@_tabularnote:nn { \exp_not:n { #1 } } }
578         }
579         { #2 }
580     }
581 }
582 }
583 }
584 {
585     \NewDocumentCommand \tabularnote { o m }
586     { \@@_err_enumitem_not_loaded: }
587 }
588 }
589 \cs_new_protected:Npn \@@_err_enumitem_not_loaded:
590 {
591     \@@_error_or_warning:n { enumitem~not~loaded }
592     \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
593 }
594 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
595 { \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`.

```

596 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
597 {

```

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).

```

598     \int_zero:N \l_tmpa_int
599     \bool_if:N \l_@@_notes_detect_duplicates_bool
600     {

```

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`.

```

601      \int_zero:N \l_tmpb_int
602      \seq_map_indexed_inline:Nn \g_@@_notes_seq
603      {
604          \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
605          \tl_if_eq:nnT { { #1 } { #2 } } { { ##2 }
606              {
607                  \tl_if_novalue:nTF { #1 }
608                      { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
609                      { \int_set:Nn \l_tmpa_int { ##1 } }
610                  \seq_map_break:
611              }
612          }
613      \int_if_zero:nF \l_tmpa_int
614      { \int_add:Nn \l_tmpa_int { \g_@@_notes_caption_int } }
615  }
616  \int_if_zero:nT \l_tmpa_int
617  {
618      \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
619      \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
620  }
621  \seq_put_right:Ne \l_@@_notes_labels_seq
622  {
623      \tl_if_novalue:nTF { #1 }
624      {
625          \@@_notes_format:n
626          {
627              \int_eval:n
628              {
629                  \int_if_zero:nTF \l_tmpa_int
630                      \c@tabularnote
631                      \l_tmpa_int
632              }
633          }
634      }
635      { #1 }
636  }
637  \peek_meaning:NF \tabularnote
638  {

```

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`.

```

639      \hbox_set:Nn \l_tmpa_box
640      {

```

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

```

641          \@@_notes_label_in_tabular:n
642          { \seq_use:Nn \l_@@_notes_labels_seq { , } }
643      }

```

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

```

644      \int_gdecr:N \c@tabularnote
645      \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.

```

646      \int_gincr:N \g_@@_tabularnote_int
647      \refstepcounter { tabularnote }
648      \int_compare:nNnT \l_tmpa_int = \c@tabularnote
649      { \int_gincr:N \c@tabularnote }
650      \seq_clear:N \l_@@_notes_labels_seq
651      \bool_lazy_or:nnTF
652      { \str_if_eq_p:ee c \l_@@_hpos_cell_tl }
653      { \str_if_eq_p:ee r \l_@@_hpos_cell_tl }
654      {
655        \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).

```

656      \skip_horizontal:n { \box_wd:N \l_tmpa_box }
657    }
658    { \box_use:N \l_tmpa_box }
659  }
660 }

```

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`.

```

661 \cs_new_protected:Npn \g_@@_tabularnote_caption:nn #1 #2
662 {
663   \bool_if:NTF \g_@@_caption_finished_bool
664   {
665     \int_compare:nNnT \c@tabularnote = \g_@@_notes_caption_int
666     { \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`!

```

667     \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
668     { \@@_error:n { Identical-notes-in-caption } }
669   }
670   {

```

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

```

671     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
672     {

```

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.

```

673       \bool_gset_true:N \g_@@_caption_finished_bool
674       \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
675       \int_gzero:N \c@tabularnote
676     }
677     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
678   }

```

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!

```

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

```



```

683         { \@@_notes_format:n { \int_use:N \c@tabularnote } }
684         { #1 }
685     }
686     \peek_meaning:NF \tabularnote
687     {
688         \@@_notes_label_in_tabular:n { \seq_use:Nn \l_@@_notes_labels_seq { , } }
689         \seq_clear:N \l_@@_notes_labels_seq
690     }
691 }
692 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
693 { \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.

```

694 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
695 {
696     \begin { pgfscope }
697     \pgfset
698     {
699         inner~sep = \c_zero_dim ,
700         minimum~size = \c_zero_dim
701     }
702     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
703     \pgfnode
704     { rectangle }
705     { center }
706     {
707         \vbox_to_ht:nn
708         { \dim_abs:n { #5 - #3 } }
709         {
710             \vfill
711             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
712         }
713     }
714     { #1 }
715     { }
716     \end { pgfscope }
717 }

```

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.

```

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

```

```

732     {
733         \vbox_to_ht:nn
734         { \dim_abs:n \l_tmpb_dim }
735         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
736     }
737     { #1 }
738     { }
739 \end { pgfscope }
740 }

```

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).

```

741 \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.

```

742 \dim_new:N \l_@@_xdots_inter_dim
743 \AtBeginDocument
744 { \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).

```

745 \dim_new:N \l_@@_xdots_shorten_start_dim
746 \dim_new:N \l_@@_xdots_shorten_end_dim
747 \AtBeginDocument
748 {
749     \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
750     \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
751 }

```

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.

```

752 \dim_new:N \l_@@_xdots_radius_dim
753 \AtBeginDocument
754 { \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.

```

755 \tl_new:N \l_@@_xdots_line_style_tl
756 \tl_const:Nn \c_@@_standard_tl { standard }
757 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl

```

The boolean `\l_@@_light_syntax_bool` corresponds to the option `light-syntax` and the boolean `\l_@@_light_syntax_expanded_bool` correspond to the the option `light-syntax-expanded`.

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

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

```
760 \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).

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

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

```
762 \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.

```
763 \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”.

```
764 \bool_new:N \l_@@_medium_nodes_bool
765 \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).

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

```
767 \keys_define:nn { nicematrix / xdots }
768 {
```

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.).

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

```

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

```

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

```

823 down .code:n = \tl\_put\_right:Nn \l_@@_xdots_down_tl { #1 } ,
824 up .code:n = \tl\_put\_right:Nn \l_@@_xdots_up_tl { #1 } ,
825 middle .code:n = \tl\_put\_right:Nn \l_@@_xdots_middle_tl { #1 } ,

```

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

```

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

```

```

830 \keys_define:nn { nicematrix / trees }
831 {
832     color.tl_set:N = \l_@@_trees_color_tl ,
833     color .value_required:n = true ,
834     line-width .dim_set:N = \l_@@_trees_line_width_dim ,
835     rounded-corners .dim_set:N = \l_@@_trees_rounded_corners_dim ,
836     rounded-corners .default:n = 2 pt ,
837     unknown .code:n =
838         \@@_unknown_key:nn { nicematrix / trees } { Unknown~key~for~trees }
839 }

```

```

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

```

```

843     \bool_set_true:N \l_@@_fix_vertex_bool
844     \socket_assign_plug:nn { nicematrix / draw-at-odd-vertex-h } { active }
845     \socket_assign_plug:nn { nicematrix / draw-at-odd-vertex-v } { active } ,
846     color .tl_set:N = \l_@@_rules_color_tl ,
847     color .value_required:n = true ,
848     width .dim_set:N = \arrayrulewidth ,
849     unknown .code:n = \@@_error:n { Unknown~key~for~rules }
850 }

```

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

```

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

```

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

```

870     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
871     rounded-corners .default:n = 4 pt ,
872     custom-line .code:n = \@@_custom_line:n { #1 } ,
873     default-line .code:n = \@@_default_line:n { #1 } ,
874     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
875     rules .value_required:n = true ,
876     trees .code:n = \keys_set:nn { nicematrix / trees } { #1 } ,
877     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`.

```

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

```

```

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

```

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

```

911     end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
912     end-of-row .value_required:n = true ,
913     end-of-row .initial:n = ; ,
914
915     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
916     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
917     last-row .int_set:N = \l_@@_last_row_int ,
918     last-row .default:n = -1 ,
919
920     code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
921     code-for-first-col .value_required:n = true ,
922     code-for-first-col+ .code:n =
923         { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
924     code-for-first-col+ .value_required:n = true ,
925     code-for-first-col~+ .meta:n = { code-for-first-col+ = #1 } ,
926
927     code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
928     code-for-last-col .value_required:n = true ,
929     code-for-last-col+ .code:n =
930         { \tl_put_right:Nn \l_@@_code_for_last_col_tl { #1 } } ,
931     code-for-last-col+ .value_required:n = true ,
932     code-for-last-col~+ .meta:n = { code-for-last-col+ = #1 } ,
933
934     code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
935     code-for-first-row .value_required:n = true ,
936     code-for-first-row+ .code:n =
937         { \tl_put_right:Nn \l_@@_code_for_first_row_tl { #1 } } ,
938     code-for-first-row+ .value_required:n = true ,
939     code-for-first-row~+ .meta:n = { code-for-first-row+ = #1 } ,
940
941     code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
942     code-for-last-row .value_required:n = true ,
943     code-for-last-row+ .code:n =
944         { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
945     code-for-last-row+ .value_required:n = true ,
946     code-for-last-row~+ .meta:n = { code-for-last-row+ = #1 } ,
947
948     hlines .clist_set:N = \l_@@_hlines_clist ,
949     hlines .default:n = all ,
950     vlines .clist_set:N = \l_@@_vlines_clist ,
951     vlines .default:n = all ,
952
953     vlines-in-sub-matrix .code:n =
954     {
955         \tl_if_single_token:nTF { #1 }
956         {

```

```

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

```

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

```

959         { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
960     }
961     { \@@_error:n { One~letter~allowed } }
962 },
963 vlines-in-sub-matrix .value_required:n = true ,
964 hvlines .code:n =
965 {
966     \bool_set_true:N \l_@@_hvlines_bool
967     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
968     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
969 },
970 hvlines .value_forbidden:n = true ,
971 hvlines-except-borders .code:n =
972 {
973     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
974     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
975     \bool_set_true:N \l_@@_hvlines_bool
976     \bool_set_true:N \l_@@_except_borders_bool
977 },
978 hlines-except-borders .value_forbidden:n = true ,
979 hlines-except-borders .code:n =
980 {
981     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
982     \bool_set_true:N \l_@@_hlines_bool
983     \bool_set_true:N \l_@@_except_borders_bool
984 },
985 hlines-except-borders .value_forbidden:n = true ,
986 parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,
987 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.

```

988     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
989     renew-dots .value_forbidden:n = true ,
990     nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
991     create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
992     create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
993     create-extra-nodes .meta:n =
994     { create-medium-nodes , create-large-nodes } ,
995     left-margin .dim_set:N = \l_@@_left_margin_dim ,
996     left-margin .default:n = \arraycolsep ,
997     right-margin .dim_set:N = \l_@@_right_margin_dim ,
998     right-margin .default:n = \arraycolsep ,
999     margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
1000     margin .default:n = \arraycolsep ,
1001     extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
1002     extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
1003     extra-margin .meta:n =
1004     { extra-left-margin = #1 , extra-right-margin = #1 } ,
1005     extra-margin .value_required:n = true ,
1006     respect-arraystretch .code:n =
1007     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
1008     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.

```

1009     pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
1010     pgf-node-code .value_required:n = true ,
1011 }

```

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

```

1012 \keys_define:nn { nicematrix / environments }
1013 {
1014   draw-trees-in-col .code:n =
1015     \clist_gput_right:Nn \g_@@_col_with_trees_clist { #1 } ,
1016   draw-trees-in-col .value_required:n = true ,
1017   create-blocks-in-col .code:n =
1018     \clist_gput_right:Nn \g_@@_cbic_clist { #1 } ,
1019   create-blocks-in-col .value_required:n = true ,
1020   corners .clist_set:N = \l_@@_corners_clist ,
1021   corners .default:n = { NW , SW , NE , SE } ,
1022   code-before .code:n =
1023     {
1024       \tl_if_empty:nF { #1 }
1025       {
1026         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1027         \bool_set_true:N \l_@@_code_before_bool
1028       }
1029     } ,
1030   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}`.

```

1031   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
1032   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
1033   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).

```

1034   baseline .tl_set:N = \l_@@_baseline_tl ,
1035   baseline .value_required:n = true ,
1036   baseline .initial:n = c ,
1037   columns-width .code:n =

```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF` (and is expandable). `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```

1038   \str_if_eq:eeTF { #1 } { auto }
1039   { \bool_set_true:N \l_@@_auto_columns_width_bool }
1040   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
1041   columns-width .value_required:n = true ,
1042   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.

```

1043   \legacy_if:nF { measuring@ }
1044   {
1045     \str_set:Ne \l_@@_name_str { #1 }
1046     \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
1047     { \@@_err_duplicate_names:n { #1 } }
1048     { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
1049   } ,
1050   name .value_required:n = true ,
1051   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
1052   code-after .value_required:n = true ,
1053 }

1054 \cs_set:Npn \@@_err_duplicate_names:n #1
1055 { \@@_error:nn { Duplicate-name } { #1 } }

1056 \keys_define:nn { nicematrix / notes }
1057 {
1058   no-print .bool_set:N = \l_@@_notes_no_print_bool ,

```



```

1059 para .bool_set:N = \l_@@_notes_para_bool ,
1060 code-before .tl_set:N = \l_@@_notes_code_before_tl ,
1061 code-before .value_required:n = true ,
1062 code-before+ .code:n =
1063   \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1064 code-before+ .value_required:n = true ,
1065 code-before~+ .code:n =
1066   \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1067 code-before~+ .value_required:n = true ,
1068 code-after .tl_set:N = \l_@@_notes_code_after_tl ,
1069 code-after .value_required:n = true ,
1070 bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
1071 style .cs_set:Np = \@@_notes_style:n #1 ,
1072 style .value_required:n = true ,
1073 label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
1074 label-in-tabular .value_required:n = true ,
1075 label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1076 label-in-list .value_required:n = true ,
1077 enumitem-keys .code:n =
1078   {
1079     \AtBeginDocument
1080     {
1081       \IfPackageLoadedT { enumitem }
1082       { \setlist* [ tabularnotes ] { #1 } }
1083     }
1084   } ,
1085 enumitem-keys .value_required:n = true ,
1086 enumitem-keys-para .code:n =
1087   {
1088     \AtBeginDocument
1089     {
1090       \IfPackageLoadedT { enumitem }
1091       { \setlist* [ tabularnotes* ] { #1 } }
1092     }
1093   } ,
1094 enumitem-keys-para .value_required:n = true ,
1095 detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1096 unknown .code:n =
1097   \@@_unknown_key:nn { nicematrix / notes } { Unknown-key-for-notes }
1098 }

```

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.

```

1099 \keys_define:nn { nicematrix / delimiters }
1100 {
1101   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1102   color .tl_set:N = \l_@@_delimiters_color_tl ,
1103   color .value_required:n = true ,
1104 }

```

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

```

1105 \keys_define:nn { nicematrix }
1106 {
1107   NiceMatrixOptions .inherit:n =
1108     { nicematrix / Global } ,
1109   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,

```

```

1110 NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1111 NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1112 NiceMatrixOptions / trees .inherit:n = nicematrix / trees ,
1113 NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1114 SubMatrix / rules .inherit:n = nicematrix / rules ,
1115 CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1116 CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1117 CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1118 NiceMatrix .inherit:n =
1119 {
1120     nicematrix / Global ,
1121     nicematrix / environments ,
1122 } ,
1123 NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1124 NiceMatrix / rules .inherit:n = nicematrix / rules ,
1125 NiceMatrix / trees .inherit:n = nicematrix / trees ,
1126 NiceTabular .inherit:n =
1127 {
1128     nicematrix / Global ,
1129     nicematrix / environments
1130 } ,
1131 NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1132 NiceTabular / rules .inherit:n = nicematrix / rules ,
1133 NiceTabular / notes .inherit:n = nicematrix / notes ,
1134 NiceTabular / trees .inherit:n = nicematrix / trees ,
1135 NiceArray .inherit:n =
1136 {
1137     nicematrix / Global ,
1138     nicematrix / environments ,
1139 } ,
1140 NiceArray / xdots .inherit:n = nicematrix / xdots ,
1141 NiceArray / rules .inherit:n = nicematrix / rules ,
1142 NiceArray / trees .inherit:n = nicematrix / trees ,
1143 pNiceArray .inherit:n =
1144 {
1145     nicematrix / Global ,
1146     nicematrix / environments ,
1147 } ,
1148 pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1149 pNiceArray / rules .inherit:n = nicematrix / rules ,
1150 pNiceArray / trees .inherit:n = nicematrix / trees
1151 }

```

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

```

1152 \keys_define:nn { nicematrix / NiceMatrixOptions }
1153 {
1154     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1155     delimiters / color .value_required:n = true ,
1156     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1157     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1158     delimiters .value_required:n = true ,
1159     width .dim_set:N = \l_@@_width_dim ,
1160     last-col .code:n =
1161         \tl_if_empty:nF { #1 }
1162         { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1163         \int_zero:N \l_@@_last_col_int ,
1164     small .bool_set:N = \l_@@_small_bool ,
1165     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.

```

1166 renew-matrix .code:n = \@@_renew_matrix: ,
1167 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}`.

```
1168     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.

```
1169     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)`.

```
1170     \str_if_eq:eeTF { #1 } { auto }
1171     { \@@_error:n { Option~auto~for~columns~width } }
1172     { \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.

```
1173     allow-duplicate-names .code:n =
1174     \cs_set:Nn \@@_err_duplicate_names:n { } ,
1175     allow-duplicate-names .value_forbidden:n = true ,
1176     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1177     notes .value_required:n = true ,
1178     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1179     sub-matrix .value_required:n = true ,
1180     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1181     matrix / columns-type .value_required:n = true ,
1182     caption-above .bool_set:N = \l_@@_caption_above_bool ,
1183     unknown .code:n =
1184     \@@_unknown_key:nn
1185     { nicematrix / Global , nicematrix / NiceMatrixOptions }
1186     { Unknown-key-for~NiceMatrixOptions }
1187 }
```

`\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.

```
1188 \NewDocumentCommand \NiceMatrixOptions { m }
1189 { \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.

```
1190 \keys_define:nn { nicematrix / NiceMatrix }
1191 {
1192     last-col .code:n = \tl_if_empty:nTF { #1 }
1193     {
1194         \bool_set_true:N \l_@@_last_col_without_value_bool
1195         \int_set:Nn \l_@@_last_col_int { -1 }
1196     }
1197     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1198     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1199     columns-type .value_required:n = true ,
1200     l .meta:n = { columns-type = l } ,
1201     r .meta:n = { columns-type = r } ,
1202     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1203     delimiters / color .value_required:n = true ,
1204     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1205     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1206     delimiters .value_required:n = true ,
1207     small .bool_set:N = \l_@@_small_bool ,
1208     small .value_forbidden:n = true ,
1209     unknown .code:n =
```

```

1210 \@@_unknown_key:nn
1211 { nicematrix / Global , nicematrix / environments , nicematrix / NiceMatrix }
1212 { Unknown-key-for-NiceMatrix }
1213 }

```

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

```

1214 \keys_define:nn { nicematrix / NiceArray }
1215 {

```

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.

```

1216 small .bool_set:N = \l_@@_small_bool ,
1217 small .value_forbidden:n = true ,
1218 last-col .code:n = \tl_if_empty:nF { #1 }
1219 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1220 \int_zero:N \l_@@_last_col_int ,
1221 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1222 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1223 unknown .code:n =
1224 \@@_unknown_key:nn
1225 { nicematrix / NiceArray , nicematrix / Global , nicematrix / environments }
1226 { Unknown-key-for-NiceArray }
1227 }

1228 \keys_define:nn { nicematrix / pNiceArray }
1229 {
1230 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1231 last-col .code:n = \tl_if_empty:nF { #1 }
1232 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1233 \int_zero:N \l_@@_last_col_int ,
1234 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1235 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1236 delimiters / color .value_required:n = true ,
1237 delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1238 delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1239 delimiters .value_required:n = true ,
1240 small .bool_set:N = \l_@@_small_bool ,
1241 small .value_forbidden:n = true ,
1242 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1243 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1244 unknown .code:n =
1245 \@@_unknown_key:nn
1246 { nicematrix / pNiceArray , nicematrix / Global , nicematrix / environments }
1247 { Unknown-key-for-NiceMatrix }
1248 }

```

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

```

1249 \keys_define:nn { nicematrix / NiceTabular }
1250 {

```

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.

```

1251 width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1252 \bool_set_true:N \l_@@_width_used_bool ,
1253 width .value_required:n = true ,
1254 notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1255 tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1256 tabularnote .value_required:n = true ,
1257 caption .tl_set:N = \l_@@_caption_tl ,
1258 caption .value_required:n = true ,

```

```

1259 short-caption .tl_set:N = \l_@@_short_caption_tl ,
1260 short-caption .value_required:n = true ,
1261 label .tl_set:N = \l_@@_label_tl ,
1262 label .value_required:n = true ,
1263 last-col .code:n = \tl_if_empty:nF { #1 }
1264 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1265 \int_zero:N \l_@@_last_col_int ,
1266 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1267 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1268 unknown .code:n =
1269 \@@_unknown_key:nn
1270 { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
1271 { Unknown-key-for-NiceTabular }
1272 }

```

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
```

```

1273 \keys_define:nn { nicematrix / CodeAfter }
1274 {
1275   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1276   delimiters / color .value_required:n = true ,
1277   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1278   rules .value_required:n = true ,
1279   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1280   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1281   sub-matrix .value_required:n = true ,
1282   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1283 }

```

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}`).

```

1284 \cs_new_protected:Npn \@@_cell_begin:
1285 {

```

`\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.

```
1286 \tl_gclear:N \g_@@_cell_after_hook_tl
```

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

```
1287 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1288 \cs_set_eq:NN \Hline \@@_Hline_in_cell:
```

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

```
1289 \int_gincr:N \c@jCol
```

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

Here is a version with the standard syntax of L3.

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

We will use a version a little more efficient.

```
1290 \if_int_compare:w \c@jCol = \c_one_int
1291 \if_int_compare:w \l_@@_first_col_int = \c_one_int
1292 \@@_begin_of_row:
1293 \fi:
1294 \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:`.

```
1295 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1296 \@@_tuning_not_tabular_begin:
1297 \@@_tuning_exterior_rows:
1298 \g_@@_row_style_tl
1299 }
1300 \cs_new_protected:Npn \@@_tuning_exterior_rows: { }
```

Here is a version with the standard syntax of L3.

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

We will use a version a little more efficient.

```
1301 \cs_new_protected:Npn \@@_tuning_first_last_row:
1302 {
1303 \if_int_compare:w \c@iRow = \c_zero_int
1304 \if_int_compare:w \c@jCol > \c_zero_int
1305 \l_@@_code_for_first_row_tl
1306 \@@_tuning_key_small: % 2026/04/06
1307 \xglobal \colorlet { nicematrix-first-row } { . }
1308 \fi:
1309 \else:
1310 \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row:
1311 \fi:
1312 }
```

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:nNtT { \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.

```

1313 \cs_new_protected:Npn \@@_tuning_last_row:
1314 {
1315   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1316     \l_@@_code_for_last_row_tl
1317     \@@_tuning_key_small: % 2026/04/06
1318     \xglobal \colorlet { nicematrix-last-row } { . }
1319   \fi:
1320 }

```

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

```

1321 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1322 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1323 {
1324   \m@th
1325   $ % $

```

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

```

1326   \@@_tuning_key_small:
1327 }
1328 \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.

```

1329 \cs_new_protected:Npn \@@_begin_of_row:
1330 {
1331   \int_gincr:N \c@iRow
1332   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1333   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1334   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1335   \pgfpicture
1336   \pgfrememberpicturepositiononpagetrue
1337   \pgfcoordinate
1338     { \@@_env: - row - \int_use:N \c@iRow - base }
1339     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1340   \str_if_empty:NF \l_@@_name_str
1341     {
1342       \pgfnodelalias
1343       { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1344       { \@@_env: - row - \int_use:N \c@iRow - base }
1345     }
1346   \endpgfpicture
1347 }

```

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

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

```

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

```

```

    { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
  }
  {
    \int_compare:nNnT { \c@iRow } = { 1 }
    {
      \dim_compare:nNnT
      { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_one_dim }
      { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
    }
  }
}

```

We will use a version a little more efficient.

```

1348 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1349 {
1350   \if_int_compare:w \c@iRow = \c_zero_int
1351     \if_dim:w \box_dp:N \l_@@_cell_box > \g_@@_dp_row_zero_dim
1352       \global \g_@@_dp_row_zero_dim = \box_dp:N \l_@@_cell_box
1353     \fi:
1354     \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_zero_dim
1355       \global \g_@@_ht_row_zero_dim = \box_ht:N \l_@@_cell_box
1356     \fi:
1357   \else:
1358     \if_int_compare:w \c@iRow = \c_one_int
1359       \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_one_dim
1360         \global \g_@@_ht_row_one_dim = \box_ht:N \l_@@_cell_box
1361       \fi:
1362     \fi:
1363   \fi:
1364 }

1365 \cs_new_protected:Npn \@@_rotate_cell_box:
1366 {
1367   \box_rotate:Nn \l_@@_cell_box
1368   { \bool_if:NTF \g_@@_rotate_minus_bool { -90 } { 90 } }
1369   \bool_if:NTF \g_@@_rotate_c_bool
1370   {
1371     \hbox_set:Nn \l_@@_cell_box
1372     {
1373       \m@th
1374       $ % $
1375       \vcenter { \box_use:N \l_@@_cell_box }
1376       $ % $
1377     }
1378   }
1379   {
1380     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1381     {
1382       \vbox_set_top:Nn \l_@@_cell_box
1383       {
1384         \vbox_to_zero:n { }
1385         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1386         \box_use:N \l_@@_cell_box
1387       }
1388     }
1389   }
1390   \bool_gset_false:N \g_@@_rotate_bool
1391   \bool_gset_false:N \g_@@_rotate_c_bool
1392   \bool_gset_false:N \g_@@_rotate_minus_bool
1393 }

```

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.

```

1394 \cs_set_protected:Npn \@@_adjust_size_box:
1395 {
1396   \if_dim:w \g_@@_blocks_wd_dim > \c_zero_dim
1397     \if_dim:w \g_@@_blocks_wd_dim > \box_wd:N \l_@@_cell_box
1398       \box_wd:N \l_@@_cell_box = \g_@@_blocks_wd_dim
1399     \fi:
1400     \global \g_@@_blocks_wd_dim = \c_zero_dim
1401   \fi:
1402   \if_dim:w \g_@@_blocks_dp_dim > \c_zero_dim
1403     \if_dim:w \g_@@_blocks_dp_dim > \box_dp:N \l_@@_cell_box
1404       \box_dp:N \l_@@_cell_box = \g_@@_blocks_dp_dim
1405     \fi
1406     \global \g_@@_blocks_dp_dim = \c_zero_dim
1407   \fi:
1408   \if_dim:w \g_@@_blocks_ht_dim > \c_zero_dim
1409     \if_dim:w \g_@@_blocks_ht_dim > \box_ht:N \l_@@_cell_box
1410       \box_ht:N \l_@@_cell_box = \g_@@_blocks_ht_dim
1411     \fi:
1412     \global \g_@@_blocks_ht_dim = \c_zero_dim
1413   \fi:
1414 }
1415 \cs_new_protected:Npn \@@_cell_end:
1416 {

```

The following command is nullified in the tabulars.

```

1417   \@@_tuning_not_tabular_end:
1418   \hbox_set_end:
1419   \@@_cell_end_i:
1420 }

```

```

\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.

```

1421 \cs_new_protected:Npn \@@_cell_end_i:
1422 {

```

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.

```

1423   \g_@@_cell_after_hook_tl
1424   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1425   \@@_adjust_size_box:
1426   \box_set_ht:Nn \l_@@_cell_box
1427   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1428   \box_set_dp:Nn \l_@@_cell_box
1429   { \box_dp:N \l_@@_cell_box + \l_@@_cell_space_bottom_limit_dim }

```

We want to compute in `\g_@@_max_cell_width_dim` the width of the widest cell of the array (except the cells of the “first column” and the “last column”).

```

1430   \@@_update_max_cell_width:

```

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

```

1431   \@@_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.

```

1432 \bool_if:NTF \g_@@_empty_cell_bool
1433 { \box_use_drop:N \l_@@_cell_box }
1434 {
1435   \bool_if:NTF \g_@@_not_empty_cell_bool
1436   \@@_print_node_cell:
1437   {
1438     \if_dim:w \box_wd:N \l_@@_cell_box > \c_zero_dim
1439     \@@_print_node_cell:
1440     \else:
1441       \box_use_drop:N \l_@@_cell_box
1442     \fi:
1443   }
1444 }
1445 \if_int_compare:w \c@jCol > \g_@@_col_total_int
1446 \global \g_@@_col_total_int = \c@jCol
1447 \fi:
1448 \global \let \g_@@_empty_cell_bool \c_false_bool
1449 \global \let \g_@@_not_empty_cell_bool \c_false_bool
1450 }

```

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:

```

1451 \cs_new_protected:Npn \@@_update_max_cell_width:
1452 {
1453   \if_dim:w \box_wd:N \l_@@_cell_box > \g_@@_max_cell_width_dim
1454   \global \g_@@_max_cell_width_dim = \box_wd:N \l_@@_cell_box
1455   \fi:
1456 }

```

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`).

```

1457 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1458 {
1459   \@@_math_toggle:
1460   \hbox_set_end:
1461   \bool_if:NF \g_@@_rotate_bool
1462   {
1463     \hbox_set:Nn \l_@@_cell_box
1464     {
1465       \makebox [ \l_@@_col_width_dim ] [ s ]
1466       { \hbox_unpack_drop:N \l_@@_cell_box }
1467     }
1468   }
1469   \@@_cell_end_i:
1470 }

```

```

1471 \pgfset
1472 {
1473   nicematrix / cell-node /.style =
1474   {
1475     inner-sep = \c_zero_dim ,
1476     minimum~width = \c_zero_dim
1477   }
1478 }

```

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.

```

1479 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1480 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1481 {
1482   \use:c
1483   {
1484     __siunitx_table_align_
1485     \bool_if:NTF \l__siunitx_table_text_bool
1486       \l__siunitx_table_align_text_tl
1487       \l__siunitx_table_align_number_str
1488     :n
1489   }
1490   { #1 }
1491 }

```

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!

```

1492 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1493 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1494 {
1495   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1496   \hbox:n
1497   {
1498     \pgfsys@markposition
1499     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1500   }
1501   #1
1502   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1503   \hbox:n
1504   {
1505     \pgfsys@markposition
1506     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1507   }
1508 }

1509 \cs_new_protected:Npn \@@_print_node_cell:
1510 {
1511   \socket_use:nn { nicematrix / siunitx-wrap }
1512   { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1513 }

```

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

```

1514 \cs_new_protected:Npn \@@_node_cell:
1515 {
1516   \pgfpicture
1517   \pgfsetbaseline \c_zero_dim
1518   \pgfrememberpicturepositiononpagetrue
1519   \pgfset { nicematrix / cell-node }
1520   \pgfnode
1521   { rectangle }
1522   { base }
1523   {

```

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).

```

1524   \sys_if_engine_xetex:T { \set@color }

```

```

1525     \box_use:N \l_@@_cell_box
1526   }
1527   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1528   { \l_@@_pgf_node_code_tl }
1529   \str_if_empty:NF \l_@@_name_str
1530   {
1531     \pgfnodealias
1532       { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1533       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1534   }
1535   \endpgfpicture
1536 }

```

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).

```

1537 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1538 {
1539   \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce
1540   { g_@@_ #2 _ lines _ tl }
1541   {
1542     \use:c { @@ _ draw _ #2 : nnn }
1543     { \int_use:N \c@iRow }
1544     { \int_use:N \c@jCol }
1545     { \exp_not:n { #3 } }
1546   }
1547 }

1548 \cs_new_protected:Npn \@@_array:n
1549 {
1550   \dim_set:Nn \col@sep
1551   { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1552   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1553   { \def \@halignto { } }
1554   { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1555   \@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).
1556   [ \str_if_eq:eeTF c \l_@@_baseline_tl c t ]
1557 }
1558 \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`.

```
1559 \cs_new_eq:cN { @@_old_ar@ialign: } \ar@ialign
```

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

```
1560 \cs_new_protected:Npn \@@_create_row_node:
1561 {
1562   \int_compare:nNt \c@iRow > \g_@@_last_row_node_int
1563   {
1564     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1565     \@@_create_row_node_i:
1566   }
1567 }

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

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

```
1570   \hbox
1571   {
1572     \bool_if:NT \l_@@_code_before_bool
1573     {
1574       \vtop
1575       {
1576         \skip_vertical:N 0.5\arrayrulewidth
1577         \pgfsys@markposition
1578         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1579         \skip_vertical:N -0.5\arrayrulewidth
1580       }
1581     }
1582     \pgfpicture
1583     \pgfrememberpicturepositiononpagetrue
1584     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1585     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1586     \str_if_empty:NF \l_@@_name_str
1587     {
1588       \pgfnodealias
1589       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1590       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1591     }
1592     \endpgfpicture
1593   }
1594 }
```

```
1595 \cs_new_protected:Npn \@@_in_everycr:
1596 {
1597   \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1598   \tbl_update_cell_data_for_next_row:
1599   \int_gzero:N \c@jCol
1600   \bool_gset_false:N \g_@@_after_col_zero_bool
1601   \bool_if:NF \g_@@_row_of_col_done_bool
1602   {
1603     \@@_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).

```
1604     \clist_if_empty:NF \l_@@_hlines_clist
1605     {
1606       \str_if_eq:eeF \l_@@_hlines_clist { all }
1607       {
1608         \clist_if_in:NtF
1609         \l_@@_hlines_clist
```

```

1610         { \int_eval:n { \c@iRow + 1 } }
1611     }
1612 {

```

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.

```

1613         \int_compare:nNnT \c@iRow > { -1 }
1614     {
1615         \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1616             { \hrule height \arrayrulewidth width \c_zero_dim }
1617     }
1618 }
1619 }
1620 }
1621 }

```

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

```

1622 \cs_set_protected:Npn \@@_renew_dots:
1623 {
1624     \cs_set_eq:NN \ldots \@@_Ldots:
1625     \cs_set_eq:NN \cdots \@@_Cdots:
1626     \cs_set_eq:NN \vdots \@@_Vdots:
1627     \cs_set_eq:NN \ddots \@@_Ddots:
1628     \cs_set_eq:NN \iddots \@@_Iddots:
1629     \cs_set_eq:NN \dots \@@_Ldots:
1630     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1631 }

```

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.

```

1632 \AtBeginDocument
1633 {
1634     \IfPackageLoadedTF { booktabs }
1635     {
1636         \cs_new_protected:Npn \@@_patch_booktabs:
1637             { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1638     }
1639     { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1640 }

```

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`.

```

1641 \cs_new_protected:Npn \@@_some_initialization:
1642 {
1643     \@@_everycr:
1644     \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1645     \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1646     \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim

```

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}`.

```

1647 \dim_gzero:N \g_@@_dp_ante_last_row_dim
1648 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1649 \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1650 }

```

`\@@_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`.

```

1651 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1652 {

```

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

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

```

1653 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1654 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1655 \seq_gclear_new:N \g_@@_multicolumn_sizes_seq

```

The command `\create_row_node:` will create a row-node (and not a row of nodes!). However, at the end of the array we construct a “false row” (for the col-nodes) and it interferes with the construction of the last row-node of the array. We don’t want to create such row-node twice (to avoid warnings or, maybe, errors). That’s why the command `\@@_create_row_node:` will use the following counter to avoid such construction.

```

1656 \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.

```

1657 \fp_gzero:N \g_@@_total_X_weight_fp
1658 \bool_gset_false:N \g_@@_V_of_X_bool

1659 \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1660 \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol

1661 \@@_patch_booktabs:
1662 \box_clear_new:N \l_@@_cell_box
1663 \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}`).

```

1664 \bool_if:NT \l_@@_small_bool
1665 {
1666     \def \arraystretch { 0.47 }
1667     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1668 \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1669 }

```

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!

```

1670 \bool_if:NT \g_@@_create_cell_nodes_bool
1671 {
1672     \tl_put_right:Nn \@@_begin_of_row:
1673     {
1674         \pgfsys@markposition

```



```

1675         { \@@_env: - row - \int_use:N \c@iRow - base }
1676     }
1677     \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1678 }

```

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`.

```

1679 \def \ar@ialign
1680 {
1681     \tbl_init_cell_data_for_table:
1682     \@@_some_initialization:
1683     \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`.

```

1684     \cs_set_eq:Nc \ar@ialign { \@@_old_ar@ialign: }
1685     \halign
1686 }

```

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).

```

1687 \cs_set_eq:NN \@@_old_ldots: \ldots
1688 \cs_set_eq:NN \@@_old_cdots: \cdots
1689 \cs_set_eq:NN \@@_old_vdots: \vdots
1690 \cs_set_eq:NN \@@_old_ddots: \ddots
1691 \cs_set_eq:NN \@@_old_iddots: \iddots
1692 \bool_if:NTF \l_@@_standard_cline_bool
1693 { \cs_set_eq:NN \cline \@@_standard_cline: }
1694 { \cs_set_eq:NN \cline \@@_cline: }
1695 \cs_set_eq:NN \Ldots \@@_Ldots:
1696 \cs_set_eq:NN \Cdots \@@_Cdots:
1697 \cs_set_eq:NN \Vdots \@@_Vdots:
1698 \cs_set_eq:NN \Ddots \@@_Ddots:
1699 \cs_set_eq:NN \Iddots \@@_Iddots:
1700 \cs_set_eq:NN \Hline \@@_Hline:
1701 \cs_set_eq:NN \Hspace \@@_Hspace:
1702 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1703 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1704 \cs_set_eq:NN \Block \@@_Block:
1705 \cs_set_eq:NN \rotate \@@_rotate:
1706 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1707 \cs_set_eq:NN \dotfill \@@_dotfill:
1708 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1709 \cs_if_free:NT \Body { \cs_set_eq:NN \Body \@@_Body: }
1710 \cs_if_free:NT \CodeBefore { \cs_set_eq:NN \CodeBefore \@@_CodeBefore: }
1711 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1712 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1713 \cs_set_eq:NN \TopRule \@@_TopRule
1714 \cs_set_eq:NN \MidRule \@@_MidRule
1715 \cs_set_eq:NN \BottomRule \@@_BottomRule
1716 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1717 \cs_set_eq:NN \Hbrace \@@_Hbrace
1718 \cs_set_eq:NN \Vbrace \@@_Vbrace
1719 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1720 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1721 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1722 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1723 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1724 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular

```

```

1725 \int_if_zero:nTF \l_@@_first_row_int
1726 { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_first_last_row: }
1727 {
1728   \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1729   { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row: }
1730 }
1731 \bool_if:NT \l_@@_renew_dots_bool { \@@_renew_dots: }

```

We redefine `\multicolumn` and, since we want `\multicolumn` to be available in the potential environments `{tabular}` nested in the environments of `nicematrix`, we patch `{tabular}` to go back to the original definition. A `\hook_gremove_code:nn` will be put in `\@@_after_array:`.

```

1732 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1733 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1734 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1735 \@@_revert_colortbl:

```

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

```

1736 \tl_if_exist:NT \l_@@_note_in_caption_tl
1737 {
1738   \tl_if_empty:NF \l_@@_note_in_caption_tl
1739   {
1740     \int_gset:Nn \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1741     \int_gset:Nn \c@tabularnote \l_@@_note_in_caption_tl
1742   }
1743 }

```

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).

```

1744 \seq_gclear:N \g_@@_multicolumn_cells_seq
1745 \seq_gclear:N \g_@@_multicolumn_sizes_seq

```

The counter `\c@iRow` will be used to count the rows of the array (its incrementation will be in the first cell of the row).

```

1746 \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`).

```

1747 \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.

```

1748 \int_gzero:N \g_@@_col_total_int
1749 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1750 \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.

```

1751 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1752 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1753 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1754 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1755 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1756 \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

1757 \tl_gclear:N \g_nicematrix_code_before_tl
1758 \tl_gclear:N \g_@@_pre_code_before_tl

```

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

```

1759 \dim_zero_new:N \l_@@_left_delim_dim
1760 \dim_zero_new:N \l_@@_right_delim_dim
1761 \bool_if:NTF \g_@@_delims_bool
1762 {

```

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

```

1763 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1764 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1765 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1766 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1767 }
1768 {
1769 \dim_gset:Nn \l_@@_left_delim_dim
1770 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1771 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1772 }
1773 }

```

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.

```

1774 \cs_new_protected:Npn \@@_pre_array:
1775 {
1776 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1777 \int_gzero_new:N \c@iRow
1778 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1779 \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).

```

1780 \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1781 { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1782 \int_compare:nNnT \l_@@_last_col_int > \c_zero_int
1783 { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1784 \bool_if:NT \g_@@_aux_found_bool
1785 {
1786 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1787 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1788 \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1789 \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1790 }

```

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).

```

1791 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1792 {
1793 \bool_set_true:N \l_@@_last_row_without_value_bool
1794 \bool_if:NT \g_@@_aux_found_bool
1795 { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1796 }
1797 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1798 {
1799 \bool_if:NT \g_@@_aux_found_bool

```

```

1800      { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1801    }

```

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”.

```

1802    \int_compare:nNtT \l_@@_last_row_int > { -2 }
1803    {
1804      \tl_put_right:Nn \@@_update_for_first_and_last_row:
1805      {
1806        \dim_compare:nNtT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1807        { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1808        \dim_compare:nNtT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1809        { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1810      }
1811    }

1812    \seq_gclear:N \g_@@_cols_vlism_seq
1813    \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1814    \bool_if:NT \l_@@_code_before_bool \@@_exec_code_before:

```

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

```

1815    \@@_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).

```

1816    \hbox_set:Nw \l_@@_the_array_box
1817    \skip_horizontal:N \l_@@_left_margin_dim % \skip_horizontal:N ?
1818    \skip_horizontal:N \l_@@_extra_left_margin_dim
1819    \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).

```

1820    \m@th
1821    $ % $
1822    \bool_if:NtF \l_@@_light_syntax_bool
1823    { \use:c { @@-light-syntax } }
1824    { \use:c { @@-normal-syntax } }
1825  }

```

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

```

1826    \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1827    {
1828      \tl_set:Nn \l_tmpa_tl { #1 }
1829      \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1830      { \@@_rescan_for_spanish:N \l_tmpa_tl }
1831      \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1832      \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...

```

1833    \@@_pre_array:
1834  }

```

9 The \CodeBefore

```

1835 \cs_new_protected_nopar:Npn \@@_Body: { \@@_fatal:n { Body~alone } }
1836 \cs_new_protected_nopar:Npn \@@_CodeBefore: { \@@_fatal:n { Bad~use~of~CodeBefore } }

```

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

```

1837 \cs_new_protected:Npn \@@_pre_code_before:
1838 {

```

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.

```

1839 \pgfsys@markposition { \@@_env: - position }
1840 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1841 \pgfpicture
1842 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1843 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1844 {
1845     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1846     \pgfcoordinate { \@@_env: - row - ##1 }
1847     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1848 }

```

Now, the recreation of the col nodes.

```

1849 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1850 {
1851     \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1852     \pgfcoordinate { \@@_env: - col - ##1 }
1853     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1854 }

```

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

```

1855 \bool_if:NT \g_@@_create_cell_nodes_bool \@@_recreate_cell_nodes:
1856 \endpgfpicture

```

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

```

1857 \@@_create_diag_nodes:

```

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

```

1858 \@@_create_blocks_nodes:
1859 \IfPackageLoadedT { tikz }
1860 {
1861     \tikzset
1862     {
1863         every-picture / .style =
1864         { overlay , name~prefix = \@@_env: - }
1865     }
1866 }
1867 \cs_set_eq:NN \cellcolor \@@_cellcolor
1868 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1869 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1870 \cs_set_eq:NN \rowcolor \@@_rowcolor
1871 \cs_set_eq:NN \rowcolors \@@_rowcolors
1872 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1873 \cs_set_eq:NN \arraycolor \@@_arraycolor
1874 \cs_set_eq:NN \columncolor \@@_columncolor
1875 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1876 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1877 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1878 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNamesCodeBefore
1879 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1880 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n

```

```

1881 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1882 }

```

```

1883 \cs_new_protected:Npn \@@_exec_code_before:
1884 {

```

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...

```

1885 \clist_map_inline:Nn \l_@@_corners_cells_clist
1886 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1887 \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`.

```

1888 \@@_add_to_colors_seq:nn { { nocolor } } { }
1889 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1890 \group_begin:

```

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

```

1891 \if_mode_math:
1892 \@@_exec_code_before_i:
1893 \else:
1894 $ % $
1895 \@@_exec_code_before_i:
1896 $ % $
1897 \fi:
1898 \group_end:
1899 }

```

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`).

```

1900 \cs_new_protected:Npn \@@_exec_code_before_i:
1901 {
1902 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1903 { \@@_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`.

```

1904 \exp_last_unbraced:No \@@_CodeBefore_keys:
1905 \g_@@_pre_code_before_tl

```

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

```

1906 \@@_actually_color:
1907 \l_@@_code_before_tl
1908 \q_stop
1909 }

```

```

1910 \keys_define:nn { nicematrix / CodeBefore }
1911 {
1912 create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1913 sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1914 sub-matrix .value_required:n = true ,
1915 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1916 delimiters / color .value_required:n = true ,
1917 unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1918 }

```

```

1919 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1920 {
1921   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1922   \@@_CodeBefore:w
1923 }

```

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.

```

1924 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1925 {
1926   \bool_if:NTF \g_@@_aux_found_bool
1927   {
1928     \@@_pre_code_before:
1929     \legacy_if:nF { measuring@ } { #1 }
1930   }

```

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).

```

1931 {
1932   \pgfsys@markposition { \@@_env: - position }
1933   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1934   \pgfpicture
1935   \pgf@relevantforpicturesizefalse
1936   \endpgfpicture

```

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

```

1937   \pgfpicture
1938   \pgfrememberpicturepositiononpagetrue
1939   \endpgfpicture

```

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

```

1940   \pgfpicture
1941   \pgf@relevantforpicturesizefalse
1942   \pgfrememberpicturepositiononpagetrue
1943   \endpgfpicture

```

The following picture corresponds `\@@_actually_color:`

```

1944   \pgfpicture
1945   \pgf@relevantforpicturesizefalse
1946   \endpgfpicture
1947 }
1948 }

```

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.

```

1949 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1950 {
1951   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1952   {
1953     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1954     \pgfcoordinate { \@@_env: - row - ##1 - base }
1955     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1956     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1957     {
1958       \cs_if_exist:cT
1959       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1960       {
1961         \pgfsys@getposition
1962         { \@@_env: - ##1 - #####1 - NW }
1963         \@@_node_position:
1964         \pgfsys@getposition

```

```

1965         { \@@_env: - ##1 - ####1 - SE }
1966         \@@_node_position_i:
1967         \@@_pgf_rect_node:nnn
1968         { \@@_env: - ##1 - ####1 }
1969         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1970         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1971     }
1972 }
1973 }
1974 \@@_create_extra_nodes:
1975 \@@_create_aliases_last:
1976 }

1977 \cs_new_protected:Npn \@@_create_aliases_last:
1978 {
1979     \int_step_inline:nn \c@iRow
1980     {
1981         \pgfnodealias
1982         { \@@_env: - ##1 - last }
1983         { \@@_env: - ##1 - \int_use:N \c@jCol }
1984     }
1985     \int_step_inline:nn \c@jCol
1986     {
1987         \pgfnodealias
1988         { \@@_env: - last - ##1 }
1989         { \@@_env: - \int_use:N \c@iRow - ##1 }
1990     }
1991     \pgfnodealias
1992     { \@@_env: - last - last }
1993     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1994 }

1995 \cs_new_protected:Npn \@@_create_blocks_nodes:
1996 {
1997     \pgfpicture
1998     \pgf@relevantforpicturesizefalse
1999     \pgfrememberpicturepositiononpagetrue
2000     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
2001     { \@@_create_one_block_node:nnnnn ##1 }
2002     \endpgfpicture
2003 }

```

The following command is called `\@@_create_one_block_node:nnnnn` but, in fact, it creates a node only if the last argument (#5) which is the name of the block, is not empty.⁷

```

2004 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
2005 {
2006     \tl_if_empty:nF { #5 }
2007     {
2008         \@@_qpoint:n { col - #2 }
2009         \dim_set_eq:NN \l_tmpa_dim \pgf@x
2010         \@@_qpoint:n { #1 }
2011         \dim_set_eq:NN \l_tmpb_dim \pgf@y
2012         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
2013         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
2014         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
2015         \@@_pgf_rect_node:nnnnn
2016         { \@@_env: - #5 }
2017         { \dim_use:N \l_tmpa_dim }
2018         { \dim_use:N \l_tmpb_dim }

```

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


```

2019         { \dim_use:N \l_@@_tmpc_dim }
2020         { \dim_use:N \pgf@y }
2021     }
2022 }

```

10 The environment `{NiceArrayWithDelims}`

```

2023 \NewDocumentEnvironment { NiceArrayWithDelims }
2024 { m m 0 { } m ! 0 { } t \CodeBefore }
2025 {

```

```

2026     \@@_provide_pgfsyspdfmark:
2027     \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 exponent to a matrix in a mathematical formula.

```

2028     \bgroup

2029     \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2030     \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2031     \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
2032     \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

```

```

2033     \int_gzero:N \g_@@_block_box_int
2034     \dim_gzero:N \g_@@_width_last_col_dim
2035     \dim_gzero:N \g_@@_width_first_col_dim
2036     \bool_gset_false:N \g_@@_row_of_col_done_bool
2037     \str_if_empty:NT \g_@@_name_env_str
2038     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
2039     \bool_if:NTF \l_@@_tabular_bool
2040     \mode_leave_vertical:
2041     \@@_test_if_math_mode:
2042     \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
2043     \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁸. This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able to use color for the rules of the array when `colortbl` is *not* loaded. That's why we do the following instruction which is in the patch of the beginning of arrays done by `colortbl`. Of course, we restore the value of `\CT@arc@` at the end of our environment.

```

2044     \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.

```

2045     \cs_if_exist:NT \tikz@library@external@loaded
2046     {
2047         \tikzexternaldisable
2048         \cs_if_exist:NT \ifstandalone
2049         { \tikzset { external / optimize = false } }
2050     }

```

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

```

2051     \int_gincr:N \g_@@_env_int
2052     \bool_if:NF \l_@@_block_auto_columns_width_bool
2053     { \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.

```

2054     \seq_gclear:N \g_@@_blocks_seq
2055     \seq_gclear:N \g_@@_pos_of_blocks_seq

```

⁸e.g. `\color[rgb]{0.5,0.5,0}`

In fact, the sequence `\g_@@_pos_of_blocks_seq` will also contain the positions of the cells with a `\diagbox` and the `\multicolumn`.

```

2056 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
2057 \seq_gclear:N \g_@@_pos_of_xdots_seq
2058 \tl_gclear_new:N \g_@@_code_before_tl
2059 \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.

```

2060 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2061 {
2062   \bool_gset_true:N \g_@@_aux_found_bool
2063   \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2064 }
2065 { \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.

```

2066 \tl_gclear:N \g_@@_aux_tl
2067 \tl_if_empty:NF \g_@@_code_before_tl
2068 {
2069   \bool_set_true:N \l_@@_code_before_bool
2070   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
2071 }
2072 \tl_if_empty:NF \g_@@_pre_code_before_tl
2073 { \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`.

```

2074 \bool_if:NTF \g_@@_delims_bool
2075 { \keys_set:nn { nicematrix / pNiceArray } }
2076 { \keys_set:nn { nicematrix / NiceArray } }
2077 { #3 , #5 }

2078 \@@_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:`.

```

2079 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
2080 }

```

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

```

2081 {
2082   \bool_if:NTF \l_@@_light_syntax_bool
2083   { \use:c { end @@-light-syntax } }
2084   { \use:c { end @@-normal-syntax } }
2085   $ % $
2086   \skip_horizontal:N \l_@@_right_margin_dim
2087   \skip_horizontal:N \l_@@_extra_right_margin_dim
2088   \hbox_set_end:
2089   \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.

```

2090 \bool_if:NT \l_@@_width_used_bool
2091 {
2092   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }

```

```

2093     { \@@_error_or_warning:n { width-without-X-columns } }
2094 }

```

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

```

2095 \fp_compare:nNnT \g_@@_total_X_weight_fp > \c_zero_fp
2096 \@@_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).

```

2097 \int_compare:nNnT \l_@@_last_row_int > { -2 }
2098 {
2099   \bool_if:NF \l_@@_last_row_without_value_bool
2100   {
2101     \int_compare:nNnF \l_@@_last_row_int = \c_iRow
2102     {
2103       \@@_error:n { Wrong-last-row }
2104       \int_set_eq:NN \l_@@_last_row_int \c_iRow
2105     }
2106   }
2107 }

```

Now, the definition of `\c@jCol` and `\g_@@_col_total_int` changes: `\c@jCol` will be the number of columns without the “last column”; `\g_@@_col_total_int` will be the number of columns with this “last column”.⁹

```

2108 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2109 \bool_if:NTF \g_@@_last_col_found_bool
2110 { \int_gdecr:N \c@jCol }
2111 {
2112   \int_compare:nNnT \l_@@_last_col_int > { -1 }
2113   { \@@_error:n { last-col-not-used } }
2114 }

```

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

```

2115 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2116 \int_compare:nNnT \l_@@_last_row_int > { -1 }
2117 { \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. 95).

```

2118 \int_if_zero:nT \l_@@_first_col_int
2119 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2120 \bool_if:nTF { ! \g_@@_delims_bool }
2121 {
2122   \str_if_eq:eeTF c \l_@@_baseline_tl
2123   \@@_use_arraybox_with_notes_c:
2124   {
2125     \str_if_eq:eeTF b \l_@@_baseline_tl
2126     \@@_use_arraybox_with_notes_b:
2127     \@@_use_arraybox_with_notes:
2128   }
2129 }

```

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).

```

2130 {
2131   \int_if_zero:nTF \l_@@_first_row_int

```

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

```

2132     {
2133         \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2134         \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2135     }
2136     { \dim_zero:N \l_tmpa_dim }

```

We compute `\l_tmpb_dim` which is the total height of the “last row” below the array (when the key `last-row` is used). A value of `-2` for `\l_@@_last_row_int` means that there is no “last row”.¹⁰

```

2137     \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2138     {
2139         \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2140         \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2141     }
2142     { \dim_zero:N \l_tmpb_dim }
2143     \hbox_set:Nn \l_tmpa_box
2144     {
2145         \m@th
2146         $ % $
2147         \@@_color:o \l_@@_delimiters_color_tl
2148         \exp_after:wN \left \g_@@_left_delim_tl
2149         \vcenter
2150     {

```

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.

```

2151         \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2152         \hbox
2153         {
2154             \bool_if:NTF \l_@@_tabular_bool
2155             { \skip_horizontal:n { - \tabcolsep } }
2156             { \skip_horizontal:n { - \arraycolsep } }
2157             \@@_use_arraybox_with_notes_c:
2158             \bool_if:NTF \l_@@_tabular_bool
2159             { \skip_horizontal:n { - \tabcolsep } }
2160             { \skip_horizontal:n { - \arraycolsep } }
2161         }

```

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

```

2162         \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2163     }
2164     \exp_after:wN \right \g_@@_right_delim_tl
2165     $ % $
2166 }

```

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.

```

2167     \bool_if:NTF \l_@@_delimiters_max_width_bool
2168     { \@@_put_box_in_flow_bis:nn \g_@@_left_delim_tl \g_@@_right_delim_tl }
2169     \@@_put_box_in_flow:
2170 }

```

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. 96).

```

2171     \bool_if:NT \g_@@_last_col_found_bool
2172     { \skip_horizontal:N \g_@@_width_last_col_dim }
2173     \bool_if:NT \l_@@_preamble_bool
2174     {
2175         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2176         { \@@_err_columns_not_used: }
2177     }
2178     \@@_after_array:

```

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

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.

```
2179 \egroup
```

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

```
2180 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2181 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2182 \iow_now:Ne \@mainaux
2183 {
2184   \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2185   \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2186   { \exp_not:o \g_@@_aux_tl }
2187 }
2188 \iow_now:Nn \@mainaux { \ExplSyntaxOff }
```

```
2189 \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2190 }
```

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

```
2191 \cs_new_protected:Npn \@@_err_columns_not_used:
2192 {
2193   \@@_warning:n { columns~not~used }
2194   \cs_gset:Npn \@@_err_columns_not_used: { }
2195 }
```

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 .

```
2196 \cs_new_protected:Npn \@@_compute_width_X:
2197 {
2198   \tl_gput_right:Ne \g_@@_aux_tl
2199   {
2200     \bool_set_true:N \l_@@_X_columns_aux_bool
2201     \dim_set:Nn \l_@@_X_columns_dim
2202     {
```

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`

```
2203     \bool_lazy_and:nnTF \g_@@_V_of_X_bool \l_@@_X_columns_aux_bool
2204     { \dim_use:N \l_@@_X_columns_dim }
2205     {
2206       \dim_compare:nNnTF
2207       {
2208         \dim_abs:n
2209         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2210       }
2211       <
2212       { 0.001 pt }
2213       { \dim_use:N \l_@@_X_columns_dim }
2214       {
2215         \dim_eval:n
2216         {
2217           \l_@@_X_columns_dim
2218           +
2219           \fp_to_dim:n
2220           {
2221             (
2222               \dim_eval:n
2223               { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2224             )
```

```

2225         / \fp_use:N \g_@@_total_X_weight_fp
2226     }
2227 }
2228 }
2229 }
2230 }
2231 }
2232 }

```

11 Construction of the preamble of the array

The final user provides a preamble, but we must convert that preamble into a preamble which will be given to `{array}` (of the package `array`).

The preamble given by the final user is stored in `\g_@@_user_preamble_tl`. The modified version will be stored in `\g_@@_array_preamble_tl`.

```

2233 \cs_new_protected:Npn \@@_transform_preamble:
2234 {
2235     \@@_transform_preamble_i:
2236     \@@_transform_preamble_ii:
2237 }
2238 \cs_new_protected:Npn \@@_transform_preamble_i:
2239 {
2240     \int_gzero:N \c@jCol

```

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

```

2241     \seq_gclear:N \g_@@_cols_vlism_seq

```

`\g_tmpb_bool` will be raised if you have a `|` at the end of the preamble provided by the final user.

```

2242     \bool_gset_false:N \g_tmpb_bool

```

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

```

2243     \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2244     \int_zero:N \l_tmpa_int
2245     \tl_gclear:N \g_@@_array_preamble_tl
2246     \str_if_eq:eeTF \l_@@_vlines_clist { all }
2247     {
2248         \tl_gset:Nn \g_@@_array_preamble_tl
2249         { ! { \skip_horizontal:N \arrayrulewidth } }
2250     }
2251     {
2252         \clist_if_in:NnT \l_@@_vlines_clist 1
2253         {
2254             \tl_gset:Nn \g_@@_array_preamble_tl
2255             { ! { \skip_horizontal:N \arrayrulewidth } }
2256         }
2257     }

```

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

```

2258     \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2259     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
2260
2261     \@@_replace_columncolor:
2262 }

```

```

2262 \cs_new_protected:Npn \@@_transform_preamble_ii:
2263 {

```

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

```

2264     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2265     {
2266         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2267         { \bool_gset_true:N \g_@@_delims_bool }
2268     }
2269     { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2270     \bool_if:NT \g_tmpb_bool { \bool_set_true:N \l_@@_bar_at_end_of_pream_bool }

```

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

```

2271     \int_if_zero:nTF \l_@@_first_col_int
2272     { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2273     {
2274         \bool_if:NF \g_@@_delims_bool
2275         {
2276             \bool_if:NF \l_@@_tabular_bool
2277             {
2278                 \clist_if_empty:NT \l_@@_vlines_clist
2279                 {
2280                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2281                     { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2282                 }
2283             }
2284         }
2285     }
2286     \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2287     { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2288     {
2289         \bool_if:NF \g_@@_delims_bool
2290         {
2291             \bool_if:NF \l_@@_tabular_bool
2292             {
2293                 \clist_if_empty:NT \l_@@_vlines_clist
2294                 {
2295                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2296                     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2297                 }
2298             }
2299         }
2300     }

```

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.

```

2301     \tag_if_active:F
2302     {

```

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

```

2303         \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2304         {
2305             \tl_gput_right:Nn \g_@@_array_preamble_tl
2306             { > { \@@_err_too_many_cols: } 1 }
2307         }
2308     }
2309 }

```

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.

```
2310 \cs_new_protected:Npn \@@_rec_preamble:n #1
2311 {
```

For the majority of the letters, we will trigger the corresponding action by calling directly a function in the main hashtable of TeX (thanks to the mechanism `\csname...\endcsname`. Be careful: all these functions take in as first argument the letter (or token) itself.¹¹

```
2312 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2313 { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2314 {
```

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

```
2315 \cs_if_exist:cTF { NC @ find @ #1 }
2316 {
2317 \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2318 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2319 }
2320 {
2321 \str_if_eq:nnTF { #1 } { S }
2322 { \@@_fatal:n { unknown~column~type~S } }
2323 { \@@_fatal:nn { unknown~column~type } { #1 } }
2324 }
2325 }
2326 }
```

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

```
2327 \cs_new_protected:Npn \@@_c: #1
2328 {
2329 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2330 \tl_gclear:N \g_@@_pre_cell_tl
2331 \tl_gput_right:Nn \g_@@_array_preamble_tl
2332 { > \@@_cell_begin: c < \@@_cell_end: }
```

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

```
2333 \int_gincr:N \c@jCol
2334 \@@_rec_preamble_after_col:n
2335 }

2336 \cs_new_protected:Npn \@@_l: #1
2337 {
2338 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2339 \tl_gclear:N \g_@@_pre_cell_tl
2340 \tl_gput_right:Nn \g_@@_array_preamble_tl
2341 {
2342 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2343 l
2344 < \@@_cell_end:
2345 }
2346 \int_gincr:N \c@jCol
2347 \@@_rec_preamble_after_col:n
2348 }
```

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


```

2349 \cs_new_protected:Npn \@@_r: #1
2350 {
2351   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2352   \tl_gclear:N \g_@@_pre_cell_tl
2353   \tl_gput_right:Nn \g_@@_array_preamble_tl
2354   {
2355     > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2356     r
2357     < \@@_cell_end:
2358   }
2359   \int_gincr:N \c@jCol
2360   \@@_rec_preamble_after_col:n
2361 }

```

For ! and @

```

2362 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2363 {
2364   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2365   \@@_rec_preamble:n
2366 }
2367 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2368 \cs_new_protected:cpn { @@ _ | : } #1
2369 {
\l_tmpa_int is the number of successive occurrences of |
2370   \int_incr:N \l_tmpa_int
2371   \@@_make_preamble_i_i:n
2372 }
2373 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2374 {

```

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

```

2375   \str_if_eq:nnTF { #1 } { | }
2376   { \use:c { @@ _ | : } | }
2377   { \@@_make_preamble_i_ii:nn { } #1 }
2378 }

```

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

```

2379 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2380 {
2381   \str_if_eq:nnTF { #2 } { [ ]
2382     { \@@_make_preamble_i_ii:nw { #1 } [ ]
2383       { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2384     }
2385   \cs_new_protected:Npn \@@_make_preamble_i_iii:nw #1 [ #2 ]
2386   { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2387 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2388 {
2389   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2390   \tl_gput_right:Ne \g_@@_array_preamble_tl
2391   {

```

Here, the command \dim_use:N is mandatory.

```

2392   \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_before_dim }
2393 }
2394 \tl_gput_right:Ne \g_@@_rules_tl
2395 {

```

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

```

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

```

We will use a version slightly more efficient:

```

2396 {
2397   \int_compare:nNt \l_tmpa_int > 1
2398   { \@@_set_multiplicity:n { \int_use:N \l_tmpa_int } }
2399   \int_set:Nn \l_@@_position_int { \int_eval:n { \c@jCol + 1 } }
2400   \dim_set:Nn \l_@@_rule_width_after_dim
2401   { \dim_use:N \l_@@_rule_width_before_dim }
2402   \@@_draw_vrule:n { #2 }
2403 }

```

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.

```

2404 }
2405 \int_zero:N \l_tmpa_int
2406 \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2407 \@@_rec_preamble:n #1
2408 }

```

```

2409 \cs_new_protected:cpn { @@_ > : } #1 #2
2410 {
2411   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2412   \@@_rec_preamble:n
2413 }
2414 \bool_new:N \l_@@_bar_at_end_of_pream_bool

```

The specifier `p` (and also the specifiers `m`, `b`, `V` and `X`) have an optional argument between square brackets for a list of *key-value* pairs. Here are the corresponding keys.

```

2415 \keys_define:nn { nicematrix / p-column }
2416 {
2417   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2418   r .value_forbidden:n = true ,
2419   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2420   c .value_forbidden:n = true ,
2421   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2422   l .value_forbidden:n = true ,
2423   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2424   S .value_forbidden:n = true ,
2425   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2426   p .value_forbidden:n = true ,
2427   t .meta:n = p ,
2428   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2429   m .value_forbidden:n = true ,
2430   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2431   b .value_forbidden:n = true
2432 }

```

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

```

2433 \cs_new_protected:Npn \@@_p: #1
2434 {
2435   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2436 \@@_make_preamble_ii_i:n
2437 }
2438 \cs_new_eq:NN \@@_b: \@@_p:
2439 \cs_new_eq:NN \@@_m: \@@_p:
2440 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2441 {
2442   \str_if_eq:nnTF { #1 } { [ ] }
2443   { \@@_make_preamble_ii_ii:w [ ] }
2444   { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2445 }
2446 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2447 { \@@_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.

```

2448 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2449 {

```

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).

```

2450 \str_set:Nn \l_@@_hpos_col_str { j }
2451 \@@_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`.

```

2452 \setlength { \l_tmpa_dim } { #2 }
2453 \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2454 }
2455 \cs_new_protected:Npn \@@_keys_p_column:n #1
2456 { \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.

```

2457 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2458 {

```

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

```

2459 \use:e
2460 {
2461   \@@_make_preamble_ii_vi:nnnnnnnn
2462   { \str_if_eq:eeTF p \l_@@_vpos_col_str { t } { b } }
2463   { #1 }
2464   {
2465     \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.

```

2466 \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2467 { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2468 {

```

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

```

2469         \def \exp_not:N \l_@@_hpos_cell_tl
2470         { \str_lowercase:f { \l_@@_hpos_col_str } }
2471     }
2472     \IfPackageLoadedTF { ragged2e }
2473     {
2474         \str_case:on \l_@@_hpos_col_str
2475         {

```

The following `\exp_not:N` are mandatory.

```

2476         c { \exp_not:N \Centering }
2477         l { \exp_not:N \RaggedRight }
2478         r { \exp_not:N \RaggedLeft }
2479     }
2480 }
2481 {
2482     \str_case:on \l_@@_hpos_col_str
2483     {
2484         c { \exp_not:N \centering }
2485         l { \exp_not:N \raggedright }
2486         r { \exp_not:N \raggedleft }
2487     }
2488 }
2489 #3
2490 }
2491 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2492 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2493 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2494 { #2 }
2495 {
2496     \str_case:onF \l_@@_hpos_col_str
2497     {
2498         { j } { c }
2499         { si } { c }
2500     }

```

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

```

2501         { \str_lowercase:f \l_@@_hpos_col_str }
2502     }
2503 }

```

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

```

2504     \int_gincr:N \c@jCol
2505     \@@_rec_preamble_after_col:n
2506 }

```

#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*.

```

2507 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2508 {
2509     \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2510     {

```

```

2511 \tl_gput_right:Nn \g_@@_array_preamble_tl
2512 { > \@@_test_if_empty_for_S: }
2513 }
2514 {
2515 \str_if_eq:eeTF { #7 } { varwidth }
2516 {
2517 \tl_gput_right:Nn \g_@@_array_preamble_tl
2518 { > \@@_test_if_empty_varwidth: }
2519 }
2520 { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2521 }
2522 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2523 \tl_gc_clear:N \g_@@_pre_cell_tl
2524 \tl_gput_right:Nn \g_@@_array_preamble_tl
2525 {
2526 > {

```

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.

```

2527 \dim_set:Nn \l_@@_col_width_dim { #2 }
2528 \@@_cell_begin:

```

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

```

2529 \use:c { #7 } [ #1 ] { #2 }

```

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

```

2530 \everypar
2531 {
2532 \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2533 \everypar { }
2534 }

```

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

```

2535 #3

```

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

```

2536 \g_@@_row_style_tl
2537 \arraybackslash
2538 #5
2539 }
2540 #8
2541 < {
2542 #6

```

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

```

2543 \@finalstrut \@arstrutbox
2544 \use:c { end #7 }

```

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

```

2545 #4
2546 \@@_cell_end:
2547 }
2548 }
2549 }

```

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

```

2550 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2551 {

```

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 `&`).

```

2552 \group_align_safe_begin:
2553 \peek_meaning:NTF &
2554 \@@_the_cell_is_empty:
2555 {
2556   \peek_meaning:NTF \\\
2557   \@@_the_cell_is_empty:
2558   {
2559     \peek_meaning:NTF \crcr
2560     \@@_the_cell_is_empty:
2561     \group_align_safe_end:
2562   }
2563 }
2564 }

```

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

```

2565 \cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
2566 {
2567   \group_align_safe_begin:
2568   \peek_meaning:NTF &
2569   \@@_the_cell_is_empty_varwidth:
2570   {
2571     \peek_meaning:NTF \\\
2572     \@@_the_cell_is_empty_varwidth:
2573     {
2574       \peek_meaning:NTF \crcr
2575       \@@_the_cell_is_empty_varwidth:
2576       \group_align_safe_end:
2577     }
2578   }
2579 }

2580 \cs_new_protected:Npn \@@_the_cell_is_empty:
2581 {
2582   \group_align_safe_end:
2583   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2584   {

```

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

```

2585   \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).

```

2586   \skip_horizontal:N \l_@@_col_width_dim
2587 }
2588 }

2589 \cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
2590 {
2591   \group_align_safe_end:
2592   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2593   { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2594 }

2595 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2596 {
2597   \peek_meaning:NT \__siunitx_table_skip:n
2598   { \bool_gset_true:N \g_@@_empty_cell_bool }
2599 }

```

The following command will be used in m-columns in order to center vertically the box. In fact, despite its name, the command does not always center the cell. Indeed, if there is only one row in the cell, it should not be centered vertically. It's not possible to know the number of rows of the

cell. However, we consider (as in `array`) that if the height of the cell is no more than the height of `\strutbox`, there is only one row.

```
2600 \cs_new_protected:Npn \l_@@_center_cell_box:
2601 {
```

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

```
2602 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2603 {
2604   \dim_compare:nNnT
2605     { \box_ht:N \l_@@_cell_box }
2606     >
```

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).

```
2607   { \box_ht:N \strutbox }
2608   {
2609     \hbox_set:Nn \l_@@_cell_box
2610     {
2611       \box_move_down:nn
2612       {
2613         ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2614           + \baselineskip ) / 2
2615       }
2616       { \box_use:N \l_@@_cell_box }
2617     }
2618   }
2619 }
2620 }
```

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

```
2621 \cs_new_protected:Npn \l_@@_V: #1 #2
2622 {
2623   \str_if_eq:nnTF { #2 } { [ ] }
2624     { \@@_make_preamble_V_i:w [ ] }
2625     { \@@_make_preamble_V_i:w [ ] { #2 } }
2626 }
2627 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2628 { \@@_make_preamble_V_ii:nn { #1 } }
2629 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2630 {
2631   \str_set:Nn \l_@@_vpos_col_str { p }
2632   \str_set:Nn \l_@@_hpos_col_str { j }
2633   \@@_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`.

```
2634   \setlength { \l_tmpa_dim } { #2 }
2635   \IfPackageLoadedTF { varwidth }
2636     { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2637     {
2638       \@@_error_or_warning:n { varwidth-not-loaded }
2639       \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2640     }
2641 }
2642 % \end{macrocode}
2643 %
2644 % \medskip
2645 % For |w| and |W|
2646 % \begin{macrocode}
2647 \cs_new_protected:Npn \l_@@_w: { \@@_make_preamble_w:nnnn { } }
```

```

2648 \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.

2649 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3
2650 {
2651   \tl_if_in:nnTF { clr } { #3 }
2652   { \@@_make_preamble_w_i:nnnn { #1 } { #2 } { #3 } }
2653   {
2654     \@@_error:nn { Invalid~argument~for~w } { #3 }
2655     \@@_make_preamble_w_i:nnnn { #1 } { #2 } { c }
2656   }
2657 }

2658 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2 #3 #4
2659 {
2660   \str_if_eq:nnTF { #3 } { s }
2661   { \@@_make_preamble_w_ii:nnnn { #1 } { #4 } }
2662   { \@@_make_preamble_w_iii:nnnn { #1 } { #2 } { #3 } { #4 } }
2663 }

```

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.

```

2664 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2
2665 {
2666   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2667   \tl_gclear:N \g_@@_pre_cell_tl
2668   \tl_gput_right:Nn \g_@@_array_preamble_tl
2669   {
2670     > {

```

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.

```

2671       \setlength { \l_@@_col_width_dim } { #2 }
2672       \@@_cell_begin:
2673       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2674     }
2675     c
2676     < {
2677       \@@_cell_end_for_w_s:
2678       #1
2679       \@@_adjust_size_box:
2680       \box_use_drop:N \l_@@_cell_box
2681     }
2682   }
2683   \int_gincr:N \c@jCol
2684   \@@_rec_preamble_after_col:n
2685 }

```

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

```

2686 \cs_new_protected:Npn \@@_make_preamble_w_iii:nnnn #1 #2 #3 #4
2687 {
2688   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2689   \tl_gclear:N \g_@@_pre_cell_tl
2690   \tl_gput_right:Nn \g_@@_array_preamble_tl
2691   {
2692     > {

```


The parameter `\l_@@_col_width_dim`, which is the width of the current column, will be available in each cell of the column. It will be used by the mono-column blocks.

We use `\setlength` in order to allow `\widthof` which is a command of `calc` (when loaded `calc` redefines `\setlength`). Of course, even if `calc` is not loaded, the following code will work with the standard version of `\setlength`.

```

2693         \setlength { \l_@@_col_width_dim } { #4 }
2694         \hbox_set:Nw \l_@@_cell_box
2695         \@@_cell_begin:
2696         \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2697     }
2698     c
2699     < {
2700         \@@_cell_end:
2701         \hbox_set_end:
2702         #1
2703         \@@_adjust_size_box:
2704         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2705     }
2706 }
```

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

```

2707     \int_gincr:N \c@jCol
2708     \@@_rec_preamble_after_col:n
2709 }

2710 \cs_new_protected:Npn \@@_special_W:
2711 {
2712     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2713     { \@@_warning:n { W~warning } }
2714 }
```

For `S` (of `siunitx`).

```

2715 \AtBeginDocument
2716 {
2717     \IfPackageLoadedT { siunitx }
2718     {
2719         \cs_new_protected:Npn \@@_S: #1 #2
2720         {
2721             \str_if_eq:nnTF { #2 } { [ ] }
2722             { \@@_make_preamble_S:w [ ] }
2723             { \@@_make_preamble_S:w [ ] { #2 } }
2724         }
2725     }
2726 }

2727 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2728 { \@@_make_preamble_S:i:n { #1 } }

2729 \cs_new_protected:Npn \@@_test_siunitx:
2730 {
2731     \IfPackageAtLeastF { siunitx } { 2026/03/26 }
2732     { \@@_fatal:n { siunitx~too-old } }
2733     \cs_gset_eq:NN \@@_test_siunitx: \prg_do_nothing:
2734 }
2735 % \end{macrocode}
2736 %
2737 % \begin{macrocode}
2738 \cs_new_protected:Npn \@@_make_preamble_S:i:n #1
2739 {
2740     \@@_test_siunitx:
2741     \tl_gput_right:Nn \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2742     \tl_gclear:N \g_@@_pre_cell_tl
2743     \tl_gput_right:Nn \g_@@_array_preamble_tl
2744     {
2745         > {
```

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).

```

2746         \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2747         \keys_set:nn { siunitx } { #1 }
2748         \@@_cell_begin:
2749         \siunitx_cell_begin:w
2750     }
2751     c
2752     <
2753     {
2754         \siunitx_cell_end:

```

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

```

2755         \tl_gput_right:Nc \g_@@_cell_after_hook_tl
2756         {
2757             \bool_if:NTF \l__siunitx_table_text_bool
2758                 \bool_set_true:N
2759                 \bool_set_false:N
2760             \l__siunitx_table_text_bool
2761         }
2762         \@@_cell_end:
2763     }
2764 }

```

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

```

2765     \int_gincr:N \c@jCol
2766     \@@_rec_preamble_after_col:n
2767 }

```

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

```

2768 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2769 {
2770     \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.

```

2771     \int_if_zero:nTF \c@jCol
2772     {
2773         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2774         {

```

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

```

2775         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2776         \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2777         \@@_rec_preamble:n #2
2778     }
2779     {
2780         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2781         \@@_make_preamble_iv:nn { #1 } { #2 }
2782     }
2783 }
2784 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2785 }
2786 \cs_set_eq:cc { @@ _ \token_to_str:N [ : } { @@ _ \token_to_str:N ( : }
2787 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2788 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2789 {
2790     \tl_gput_right:Nc \g_@@_pre_code_after_tl
2791     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } } \c_true_bool }

```

```

2792 \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2793 {
2794   \@@_error:nn { delimiter~after~opening } { #2 }
2795   \@@_rec_preamble:n
2796 }
2797 { \@@_rec_preamble:n #2 }
2798 }

```

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

```

2799 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2800 { \use:c { @@ _ \token_to_str:N ( : } }

```

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

```

2801 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2802 {
2803   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2804   \tl_if_in:nnTF { ) ] \} } { #2 }
2805   { \@@_make_preamble_v:nnn #1 #2 }
2806   {
2807     \str_if_eq:nnTF \s_stop { #2 }
2808     {
2809       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2810       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2811       {
2812         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2813         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2814         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2815         \@@_rec_preamble:n #2
2816       }
2817     }
2818     {
2819       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2820       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2821       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2822       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2823       \@@_rec_preamble:n #2
2824     }
2825   }
2826 }
2827 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2828 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2829 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2830 {
2831   \str_if_eq:nnTF \s_stop { #3 }
2832   {
2833     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2834     {
2835       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2836       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2837       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2838       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2839     }
2840     {
2841       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2842       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2843       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2844       \@@_error:nn { double~closing~delimiter } { #2 }
2845     }
2846   }
2847   {

```

```

2848 \tl_gput_right:Nn \g_@@_pre_code_after_tl
2849 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2850 \@@_error:nn { double~closing~delimiter } { #2 }
2851 \@@_rec_preamble:n #3
2852 }
2853 }

2854 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2855 { \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 @{...}.

```

2856 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2857 {
2858   \str_if_eq:nnTF { #1 } { < }
2859   { \@@_rec_preamble_after_col_i:n }
2860   {
2861     \str_if_eq:nnTF { #1 } { @ }
2862     { \@@_rec_preamble_after_col_ii:n }
2863     {
2864       \str_if_eq:eeTF \l_@@_vlines_clist { all }
2865       {
2866         \tl_gput_right:Nn \g_@@_array_preamble_tl
2867         { ! { \skip_horizontal:N \arrayrulewidth } }
2868       }
2869       {
2870         \clist_if_in:NnT \l_@@_vlines_clist
2871         { \int_eval:n { \c@jCol + 1 } }
2872         {
2873           \tl_gput_right:Nn \g_@@_array_preamble_tl
2874           { ! { \skip_horizontal:N \arrayrulewidth } }
2875         }
2876       }
2877       \@@_rec_preamble:n { #1 }
2878     }
2879   }
2880 }

2881 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2882 {
2883   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2884   \@@_rec_preamble_after_col:n
2885 }

```

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.

```

2886 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2887 {
2888   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2889   {
2890     \tl_gput_right:Nn \g_@@_array_preamble_tl
2891     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2892   }
2893   {
2894     \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2895     {
2896       \tl_gput_right:Nn \g_@@_array_preamble_tl
2897       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2898     }
2899     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2900   }
2901   \@@_rec_preamble:n
2902 }

```

```

2903 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2904 {
2905   \tl_clear:N \l_tmpa_tl
2906   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2907   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2908 }

```

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

```

2909 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2910 { \@@_rec_preamble:n }

```

For the case of a letter `X`. This specifier may take in an optional argument (between square brackets). That's why we test whether there is a `[` after the letter `X`.

```

2911 \cs_new_protected:Npn \@@_X: #1 #2
2912 {
2913   \str_if_eq:nnTF { #2 } { [ ]
2914     { \@@_make_preamble_X:w [ ] }
2915     { \@@_make_preamble_X:w [ ] #2 }
2916   }
2917   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2918   { \@@_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`.

```

2919 \keys_define:nn { nicematrix / X-column }
2920 {
2921   V .code:n =
2922     \IfPackageLoadedTF { varwidth }
2923     {
2924       \bool_set_true:N \l_@@_V_of_X_bool
2925       \bool_gset_true:N \g_@@_V_of_X_bool
2926     }
2927     { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
2928   unknown .code:n =
2929     \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2930     { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2931     { \@@_error_or_warning:n { invalid~weight } }
2932 }

```

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

```

2933 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2934 {

```

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`).

```

2935   \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`).

```

2936   \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`).

```

2937   \fp_set:Nn \l_tmpa_fp { 1.0 }
2938   \@@_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.

```
2939 \bool_set_false:N \l_@@_V_of_X_bool
2940 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
```

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

```
2941 \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).

```
2942 \bool_if:NTF \l_@@_X_columns_aux_bool
2943 {
2944   \@@_make_preamble_ii_iv:nnn
```

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

```
2945 { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2946 { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
2947 { \@@_no_update_width: }
2948 }
```

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.

```
2949 {
2950   \tl_gput_right:Nn \g_@@_array_preamble_tl
2951   {
2952     > {
2953       \@@_cell_begin:
2954       \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`.

```
2955 \NotEmpty
```

The following code will nullify the box of the cell.

```
2956 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2957 { \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`.

```
2958 \begin { minipage } { 5 cm } \arraybackslash
2959 }
2960 c
2961 < {
2962   \end { minipage }
2963   \@@_cell_end:
2964 }
2965 }
2966 \int_gincr:N \c@jCol
2967 \@@_rec_preamble_after_col:n
2968 }
2969 }
```

```
2970 \cs_new_protected:Npn \@@_no_update_width:
2971 {
2972   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2973   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2974 }
```

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

```

2975 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2976 {
2977   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2978   { \int_eval:n { \c@jCol + 1 } }
2979   \tl_gput_right:Ne \g_@@_array_preamble_tl
2980   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2981   \@@_rec_preamble:n
2982 }

```

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.

```

2983 \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).

```

2984 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2985 { \@@_fatal:n { Preamble-forgotten } }
2986 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
2987 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
2988 { @@ _ \token_to_str:N \hline : }
2989 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
2990 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
2991 { @@ _ \token_to_str:N \hline : }
2992 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
2993 { @@ _ \token_to_str:N \hline : }
2994 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
2995 { @@ _ \token_to_str:N \hline : }
2996 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
2997 { @@ _ \token_to_str:N \hline : }
2998 \cs_new_protected:cpn { @@ _ \token_to_str:N \linewidth : }
2999 { \@@_fatal:n { NiceTabularX~probably~required } }
3000 \cs_set_eq:cc { @@ _ \token_to_str:N \textwidth : }
3001 { @@ _ \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).

```

3002 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
3003 {

```

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`.

```

3004   \multispan { #1 }
3005   \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
3006   \begingroup
3007   \tbl_update_multicolumn_cell_data:n { #1 }

```

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

```

3008   \tl_gclear:N \g_@@_preamble_tl
3009   \@@_make_m_preamble:n #2 \q_stop

```

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

```

3010   \def \addamp
3011   {
3012     \legacy_if:nTF { @firstamp }
3013     { \legacy_if_set_false:n { @firstamp } }

```

```

3014         { \@preamerr 5 }
3015     }
3016     \exp_args:No \@mkpream \g_@@_preamble_tl
3017     \@addtopreamble \@empty
3018     \endgroup
3019     \UseTaggingSocket { tbl / colspan } { #1 }

```

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

```

3020     \int_compare:nNtT { #1 } > 1
3021     {
3022         \seq_gput_right:Ne \g_@@_multicolumn_cells_seq
3023         { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
3024         \seq_gput_right:Nn \g_@@_multicolumn_sizes_seq { #1 }
3025         \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
3026         {
3027             {
3028                 \int_if_zero:nTF \c@jCol
3029                 { \int_eval:n { \c@iRow + 1 } }
3030                 { \int_use:N \c@iRow }
3031             }
3032             { \int_eval:n { \c@jCol + 1 } }
3033             {
3034                 \int_if_zero:nTF \c@jCol
3035                 { \int_eval:n { \c@iRow + 1 } }
3036                 { \int_use:N \c@iRow }
3037             }
3038             { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block.

```

3039         { }
3040     }
3041 }

```

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

```

3042     \RenewDocumentCommand { \cellcolor } { 0 { } m }
3043     {
3044         \tl_gput_right:Ne \g_@@_pre_code_before_tl
3045         {
3046             \@@_rectanglecolor [ ##1 ]
3047             { \exp_not:n { ##2 } }
3048             { \int_use:N \c@iRow - \int_use:N \c@jCol }
3049             { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
3050         }
3051         \ignorespaces
3052     }

```

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

```

3053     \def \@sharp { #3 }
3054     \@arstrut
3055     \@preamble
3056     \null

```

We add some lines.

```

3057     \int_gadd:Nn \c@jCol { #1 - 1 }
3058     \int_compare:nNtT \c@jCol > \g_@@_col_total_int
3059     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3060     \ignorespaces
3061 }

```


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`.

```

3062 \cs_new_protected:Npn \@@_make_m_preamble:n #1
3063 {
3064   \str_case:nnF { #1 }
3065   {
3066     c { \@@_make_m_preamble_i:n #1 }
3067     l { \@@_make_m_preamble_i:n #1 }
3068     r { \@@_make_m_preamble_i:n #1 }
3069     > { \@@_make_m_preamble_ii:nn #1 }
3070     ! { \@@_make_m_preamble_ii:nn #1 }
3071     @ { \@@_make_m_preamble_ii:nn #1 }
3072     | { \@@_make_m_preamble_iii:n #1 }
3073     p { \@@_make_m_preamble_iv:nnn t #1 }
3074     m { \@@_make_m_preamble_iv:nnn c #1 }
3075     b { \@@_make_m_preamble_iv:nnn b #1 }
3076     w { \@@_make_m_preamble_v:nnnn { } #1 }
3077     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
3078     \q_stop { }
3079   }
3080   {
3081     \cs_if_exist:cTF { NC @ find @ #1 }
3082     {
3083       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
3084       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
3085     }
3086     {
3087       \str_if_eq:nnTF { #1 } { S }
3088       { \@@_fatal:n { unknown~column~type~S~multicolumn } }
3089       { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
3090     }
3091   }
3092 }

```

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

```

3093 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
3094 {
3095   \tl_gput_right:Nn \g_@@_preamble_tl
3096   {
3097     > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
3098     #1
3099     < \@@_cell_end:
3100   }

```

We test for the presence of a `<`.

```

3101   \@@_make_m_preamble_x:n
3102 }

```

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

```

3103 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
3104 {
3105   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
3106   \@@_make_m_preamble:n
3107 }

```

For `|`

```

3108 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
3109 {
3110   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
3111   \@@_make_m_preamble:n
3112 }

```

For p, m and b

```

3113 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
3114 {
3115   \tl_gput_right:Nn \g_@@_preamble_tl
3116   {
3117     > {
3118       \@@_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`.

```

3119       \setlength { \l_tmpa_dim } { #3 }
3120       \begin { minipage } [ #1 ] { \l_tmpa_dim }
3121       \mode_leave_vertical:
3122       \arraybackslash
3123       \vrule height \box_ht:N \@@arstrutbox depth \c_zero_dim width \c_zero_dim
3124     }
3125     c
3126     < {
3127       \vrule height \c_zero_dim depth \box_dp:N \@@arstrutbox width \c_zero_dim
3128       \end { minipage }
3129       \@@_cell_end:
3130     }
3131   }

```

We test for the presence of a `<`.

```

3132   \@@_make_m_preamble_x:n
3133 }

```

For w and W

```

3134 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3135 {
3136   \tl_gput_right:Nn \g_@@_preamble_tl
3137   {
3138     > {
3139       \dim_set:Nn \l_@@_col_width_dim { #4 }
3140       \hbox_set:Nw \l_@@_cell_box
3141       \@@_cell_begin:
3142       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3143     }
3144     c
3145     < {
3146       \@@_cell_end:
3147       \hbox_set_end:
3148       \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3149       #1
3150       \@@_adjust_size_box:
3151       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3152     }
3153   }

```

We test for the presence of a `<`.

```

3154   \@@_make_m_preamble_x:n
3155 }

```

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

```

3156 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3157 {
3158   \str_if_eq:nnTF { #1 } { < }
3159   \@@_make_m_preamble_ix:n
3160   { \@@_make_m_preamble:n { #1 } }
3161 }

```

```

3162 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3163 {
3164   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3165   \@@_make_m_preamble_x:n
3166 }

```

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`).

```

3167 \cs_new_protected:Npn \@@_put_box_in_flow:
3168 {
3169   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3170   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3171   \str_if_eq:eeTF \l_@@_baseline_tl { c }
3172     { \box_use_drop:N \l_tmpa_box }
3173   \@@_put_box_in_flow_i:
3174 }

```

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

```

3175 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3176 {
3177   \pgfpicture
3178     \@@_qpoint:n { row - 1 }
3179     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3180     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3181     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3182     \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).

```

3183   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3184   {
3185     \int_set:Nn \l_tmpa_int
3186       { \str_range:Nnn \l_@@_baseline_tl { 6 } { -1 } }
3187     \bool_lazy_or:nnT
3188       { \int_compare_p:nNn \l_tmpa_int < { 1 } }
3189       { \int_compare_p:nNn \l_tmpa_int > { \c@iRow + 1 } }
3190     {
3191       \@@_error:n { bad-value-for~baseline-line }
3192       \int_set:Nn \l_tmpa_int 1
3193     }
3194     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3195   }
3196   {
3197     \str_if_eq:eeTF t \l_@@_baseline_tl
3198       { \int_set:Nn \l_tmpa_int 1 }
3199       {
3200         \str_if_eq:eeTF b \l_@@_baseline_tl
3201           { \int_set_eq:NN \l_tmpa_int \c@iRow }
3202           { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3203       }
3204     \bool_lazy_or:nnT
3205       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3206       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3207     {
3208       \@@_error:n { bad-value-for~baseline }
3209       \int_set:Nn \l_tmpa_int 1
3210     }
3211     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3212         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3213     }
3214     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3215     \endpgfpicture
3216     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3217     % \box_use_drop:N \l_tmpa_box   % 2026/04/13
3218 }

```

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).

```

3219 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3220 {

```

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.

```

3221     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3222     {
3223         \int_compare:nNnT \c@jCol > 1
3224         {
3225             \box_set_wd:Nn \l_@@_the_array_box
3226             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3227         }
3228     }

```

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

```

3229     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3230     \bool_if:NT \l_@@_caption_above_bool
3231     {
3232         \tl_if_empty:NF \l_@@_caption_tl
3233         {
3234             \bool_set_false:N \g_@@_caption_finished_bool
3235             \int_gzero:N \c@tabularnote
3236             \@@_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).

```

3237         \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3238         {
3239             \tl_gput_right:Ne \g_@@_aux_tl
3240             {
3241                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3242                 { \int_use:N \g_@@_notes_caption_int }
3243             }
3244             \int_gzero:N \g_@@_notes_caption_int
3245         }
3246     }
3247 }

```

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

```

3248     \hbox
3249     {
3250         \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.

```

3251     \@@_create_extra_nodes:
3252     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3253 }

```

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).

```

3254     \bool_lazy_any:nT
3255     {
3256       { ! \seq_if_empty_p:N \g_@@_notes_seq }
3257       { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3258       { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3259     }
3260     {
3261       \bool_if:NTF \l_@@_notes_no_print_bool
3262       { \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes: }
3263       \@@_tabular_notes:
3264     }
3265     \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3266     \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3267   \end { minipage }
3268 }

```

```

3269 \cs_new_protected:Npn \@@_insert_caption:
3270 {
3271   \tl_if_empty:NF \l_@@_caption_tl
3272   {
3273     \cs_if_exist:NTF \@capytype
3274     { \@@_insert_caption_i: }
3275     { \@@_error:n { caption~outside~float } }
3276   }
3277 }

```

```

3278 \cs_new_protected:Npn \@@_insert_caption_i:
3279 {
3280   \group_begin:

```

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

```

3281     \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.

```

3282     \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
3283     \tl_if_empty:NTF \l_@@_short_caption_tl
3284     \caption
3285     { \caption [ \l_@@_short_caption_tl ] }
3286     { \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.

```

3287     \bool_if:NF \g_@@_caption_finished_bool
3288     {
3289       \bool_gset_true:N \g_@@_caption_finished_bool

```

```

3290     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3291     \int_gzero:N \c@tabularnote
3292   }
3293   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3294   \group_end:
3295 }
3296 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3297 {
3298   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3299   \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3300 }
3301 \cs_set_protected:Npn \@@_tabular_notes_error:
3302 { \@@_error:n { Bad~use~of~NiceTabularNotes } }
3303 \cs_set_eq:NN \NiceTabularNotes \@@_tabular_notes_error:
3304 \cs_set_protected:Npn \@@_tabular_notes:
3305 {
3306   \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes_error:
3307   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3308   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3309   \skip_vertical:N 0.65ex

```

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

```

3310   \group_begin:
3311   \l_@@_notes_code_before_tl
3312   \tl_if_empty:NF \g_@@_tabularnote_tl
3313   {
3314     \g_@@_tabularnote_tl \par
3315     \tl_gclear:N \g_@@_tabularnote_tl
3316   }

```

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.

```

3317   \int_compare:nNnT \c@tabularnote > \c_zero_int
3318   {
3319     \bool_if:NTF \l_@@_notes_para_bool
3320     {
3321       \begin { tabularnotes* }
3322       \seq_map_inline:Nn \g_@@_notes_seq
3323       { \@@_one_tabularnote:nn ##1 }
3324       \strut
3325       \end { tabularnotes* }

```

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

```

3326     \par
3327   }
3328   {
3329     \tabularnotes
3330     \seq_map_inline:Nn \g_@@_notes_seq
3331     { \@@_one_tabularnote:nn ##1 }
3332     \strut
3333     \endtabularnotes
3334   }
3335 }
3336 \unskip
3337 \group_end:
3338 \bool_if:NT \l_@@_notes_bottomrule_bool
3339 {
3340   \IfPackageLoadedTF { booktabs }
3341   {

```

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

```

3342     \skip_vertical:N \aboverulesep

```

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

```

3343         { \CT@arc@ \hrule height \heavyrulewidth }
3344     }
3345     { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3346 }
3347 \l_@@_notes_code_after_tl
3348 \seq_gclear:N \g_@@_notes_seq
3349 \seq_gclear:N \g_@@_notes_in_caption_seq
3350 \int_gzero:N \c@tabularnote
3351 }

```

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

```

3352 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3353 {
3354     \tl_if_novalue:nTF { #1 }
3355     { \item }
3356     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3357 }

```

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.

```

3358 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3359 {
3360     \pgfpicture
3361     \@@_qpoint:n { row - 1 }
3362     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3363     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3364     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3365     \endpgfpicture
3366     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3367     \int_if_zero:nT \l_@@_first_row_int
3368     {
3369         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3370         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3371     }
3372     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3373 }

```

Now, the general case.

```

3374 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3375 {

```

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

```

3376     \str_if_eq:eeT \l_@@_baseline_tl { t }
3377     { \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`.

```

3378     \pgfpicture
3379     \@@_qpoint:n { row - 1 }
3380     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3381     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3382     {
3383         \int_set:Nn \l_tmpa_int
3384         {
3385             \str_range:Nnn
3386             \l_@@_baseline_tl
3387             { 6 }
3388             { \tl_count:o \l_@@_baseline_tl }

```

```

3389     }
3390     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3391 }
3392 {
3393     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3394     \bool_lazy_or:nnT
3395     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3396     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3397     {
3398         \@@_error:n { bad-value-for-baseline }
3399         \int_set:Nn \l_tmpa_int 1
3400     }
3401     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3402 }
3403 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3404 \endpgfpicture
3405 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3406 \int_if_zero:nT \l_@@_first_row_int
3407 {
3408     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3409     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3410 }
3411 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3412 }

```

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.

```

3413 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3414 {

```

We will compute the real width of both delimiters used.

```

3415     \dim_zero_new:N \l_@@_real_left_delim_dim
3416     \dim_zero_new:N \l_@@_real_right_delim_dim
3417     \hbox_set:Nn \l_tmpb_box
3418     {
3419         \m@th
3420         $ % $
3421         \left #1
3422         \vcenter
3423         {
3424             \vbox_to_ht:nn
3425             { \box_ht_plus_dp:N \l_tmpa_box }
3426             { }
3427         }
3428         \right .
3429         $ % $
3430     }
3431     \dim_set:Nn \l_@@_real_left_delim_dim
3432     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3433     \hbox_set:Nn \l_tmpb_box
3434     {
3435         \m@th
3436         $ % $
3437         \left .
3438         \vbox_to_ht:nn
3439         { \box_ht_plus_dp:N \l_tmpa_box }
3440         { }
3441         \right #2
3442         $ % $
3443     }
3444     \dim_set:Nn \l_@@_real_right_delim_dim
3445     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```


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

```

3446 \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3447 \@@_put_box_in_flow:
3448 \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3449 }

```

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).

```

3450 \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).

```

3451 {
3452   \peek_remove_spaces:n
3453   {
3454     \peek_meaning:NTF \end
3455     \@@_analyze_end:Nn
3456     {
3457       \@@_transform_preamble:
3458       \@@_array:o \g_@@_array_preamble_tl
3459     }
3460   }
3461 }
3462 {
3463   \@@_create_col_nodes:
3464   \endarray
3465 }

```

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

```

3466 \NewDocumentEnvironment { @@-light-syntax } { b }
3467 {

```

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`.

```

3468 \tl_if_empty:nT { #1 }
3469 { \@@_fatal:n { empty-environment } }
3470 \tl_if_in:nnT { #1 } { & }
3471 { \@@_fatal:n { ampersand-in-light-syntax } }
3472 \tl_if_in:nnT { #1 } { \ }
3473 { \@@_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`.

```

3474 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3475 }

```

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.

```

3476 {
3477   \@@_create_col_nodes:
3478   \endarray
3479 }

```

```

3480 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3481 {
3482   \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).

```

3483   \seq_clear_new:N \l_@@_rows_seq

```

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

```

3484   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3485   \bool_if:NTF \l_@@_light_syntax_expanded_bool
3486     \seq_set_split:Nee
3487     \seq_set_split:Non
3488     \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3489   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3490   \tl_if_empty:NF \l_tmpa_tl
3491     { \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.

```

3492   \int_compare:nNnT \l_@@_last_row_int = { -1 }
3493     { \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`).

```

3494   \tl_build_begin:N \l_@@_new_body_tl
3495   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3496   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3497   \@@_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).

```

3498   \seq_map_inline:Nn \l_@@_rows_seq
3499   {
3500     \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3501     \@@_line_with_light_syntax:n { #1 }
3502   }
3503   \tl_build_end:N \l_@@_new_body_tl
3504   \int_compare:nNnT \l_@@_last_col_int = { -1 }
3505   {
3506     \int_set:Nn \l_@@_last_col_int
3507     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3508   }

```

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.

```

3509   \@@_transform_preamble:

```

```

3510   \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3511 }

```

```

3512 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3513 {
3514   \seq_clear_new:N \l_@@_cells_seq
3515   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3516   \int_set:Nn \l_@@_nb_cols_int
3517   { \int_max:nn \l_@@_nb_cols_int { \seq_count:N \l_@@_cells_seq } }
3518   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3519   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3520   \seq_map_inline:Nn \l_@@_cells_seq

```

```

3521     { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3522   }
3523   \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`.

```

3524   \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3525   {
3526     \str_if_eq:eeT \g_@@_name_env_str { #2 }
3527     { \@@_fatal:n { empty~environment } }

```

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

```

3528     \end { #2 }
3529   }

```

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`).

```

3530   \cs_new:Npn \@@_create_col_nodes:
3531   {
3532     \crrc
3533     \int_if_zero:nT \l_@@_first_col_int
3534     {
3535       \omit
3536       \hbox_overlap_left:n
3537       {
3538         \bool_if:NT \l_@@_code_before_bool
3539         { \pgfsys@markposition { \@@_env: - col - 0 } }
3540         \pgfpicture
3541         \pgfrememberpicturepositiononpagetrue
3542         \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3543         \str_if_empty:NF \l_@@_name_str
3544         { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3545         \endpgfpicture
3546         \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3547       }
3548     }
3549   }
3550   \omit

```

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

```

3551   \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`).

```

3552   \int_if_zero:nTF \l_@@_first_col_int
3553   {
3554     \@@_mark_position:n { 1 }
3555     \pgfpicture
3556     \pgfrememberpicturepositiononpagetrue
3557     \pgfcoordinate { \@@_env: - col - 1 }
3558     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3559     \str_if_empty:NF \l_@@_name_str
3560     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3561     \endpgfpicture
3562   }
3563   {
3564     \bool_if:NT \l_@@_code_before_bool
3565     {
3566       \hbox
3567       {

```

```

3568         \skip_horizontal:n { 0.5 \arrayrulewidth }
3569         \pgfsys@markposition { \@@_env: - col - 1 }
3570         \skip_horizontal:n { -0.5 \arrayrulewidth }
3571     }
3572 }
3573 \pgfpicture
3574 \pgfrememberpicturepositiononpagetrue
3575 \pgfcoordinate { \@@_env: - col - 1 }
3576 { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3577 \@@_node_alias:n { 1 }
3578 \endpgfpicture
3579 }

```

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.

```

3580 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3581 \bool_if:NF \l_@@_auto_columns_width_bool
3582 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3583 {
3584     \bool_lazy_and:nnTF
3585     \l_@@_auto_columns_width_bool
3586     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3587     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3588     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3589     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3590 }
3591 \skip_horizontal:N \g_tmpa_skip
3592 \hbox
3593 {
3594     \@@_mark_position:n { 2 }
3595     \pgfpicture
3596     \pgfrememberpicturepositiononpagetrue
3597     \pgfcoordinate { \@@_env: - col - 2 }
3598     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3599     \@@_node_alias:n { 2 }
3600     \endpgfpicture
3601 }

```

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.

```

3602 \int_gset:Nn \g_tmpa_int 1
3603 \bool_if:NTF \g_@@_last_col_found_bool
3604 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3605 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3606 {
3607     &
3608     \omit
3609     \int_gincr:N \g_tmpa_int

```

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

```

3610     \skip_horizontal:N \g_tmpa_skip
3611     \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

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

```

3612 \pgfpicture
3613 \pgfrememberpicturepositiononpagetrue
3614 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3615 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3616 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3617 \endpgfpicture
3618 }

```

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).

```

3619 \bool_lazy_or:nnF
3620 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3621 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3622 {
3623   &
3624   \omit
3625   \skip_horizontal:N \g_tmpa_skip
3626   \int_gincr:N \g_tmpa_int
3627   \bool_lazy_any:nF
3628   {
3629     \g_@@_delims_bool
3630     \l_@@_tabular_bool
3631     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3632     \l_@@_exterior_arraycolsep_bool
3633     \l_@@_bar_at_end_of_pream_bool
3634   }
3635   { \skip_horizontal:n { - \col@sep } }
3636   \bool_if:NT \l_@@_code_before_bool
3637   {
3638     \hbox
3639     {
3640       \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.

```

3641 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3642 { \skip_horizontal:n { - \arraycolsep } }
3643 \pgfsys@markposition
3644 { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3645 \skip_horizontal:n { 0.5 \arrayrulewidth }
3646 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3647 { \skip_horizontal:N \arraycolsep }
3648 }
3649 }
3650 \pgfpicture
3651 \pgfrememberpicturepositiononpagetrue
3652 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3653 {
3654   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3655   {
3656     \pgfpoint
3657     { - 0.5 \arrayrulewidth - \arraycolsep }
3658     \c_zero_dim
3659   }
3660   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3661 }
3662 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3663 \endpgfpicture
3664 }

3665 \bool_if:NT \g_@@_last_col_found_bool
3666 {
3667   \hbox_overlap_right:n
3668   {
3669     \skip_horizontal:N \g_@@_width_last_col_dim
3670     \skip_horizontal:N \col@sep
3671     \bool_if:NT \l_@@_code_before_bool
3672     {
3673       \pgfsys@markposition
3674       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }

```

```

3675     }
3676     \pgfpicture
3677     \pgfrememberpicturepositiononpagetrue
3678     \pgfcoordinate
3679     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3680     \pgfpintorigin
3681     \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3682     \endpgfpicture
3683   }
3684 }
3685 }

3686 \cs_new_protected:Npn \@@_mark_position:n #1
3687 {
3688   \bool_if:NT \l_@@_code_before_bool
3689   {
3690     \hbox
3691     {
3692       \skip_horizontal:n { -0.5 \arrayrulewidth }
3693       \pgfsys@markposition { \@@_env: - col - #1 }
3694       \skip_horizontal:n { 0.5 \arrayrulewidth }
3695     }
3696   }
3697 }

3698 \cs_new_protected:Npn \@@_node_alias:n #1
3699 {
3700   \str_if_empty:NF \l_@@_name_str
3701   { \pgfnodealias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3702 }

```

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

```

3703 \tl_const:Nn \c_@@_preamble_first_col_tl
3704 {
3705   >
3706   {

```

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

```

3707     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3708     \bool_gset_true:N \g_@@_after_col_zero_bool
3709     \@@_begin_of_row:
3710     \hbox_set:Nw \l_@@_cell_box
3711     \@@_math_toggle:
3712     \@@_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”.

```

3713     \int_compare:nNnT \c@iRow > \c_zero_int
3714     {
3715       \bool_lazy_or:nnT
3716       { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3717       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3718       {
3719         \l_@@_code_for_first_col_tl
3720         \xglobal \colorlet { nicematrix-first-col } { . }
3721       }
3722     }
3723 }

```

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`.

```

3724   l
3725   <
3726   {
3727     \@@_math_toggle:
3728     \hbox_set_end:
3729     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3730     \@@_adjust_size_box:
3731     \@@_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.

```

3732     \dim_gset:Nn \g_@@_width_first_col_dim
3733     { \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.

```

3734     \hbox_overlap_left:n
3735     {
3736       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3737       \@@_node_cell:
3738       { \box_use_drop:N \l_@@_cell_box }
3739       \skip_horizontal:N \l_@@_left_delim_dim
3740       \skip_horizontal:N \l_@@_left_margin_dim
3741       \skip_horizontal:N \l_@@_extra_left_margin_dim
3742     }
3743     \bool_gset_false:N \g_@@_empty_cell_bool
3744     \skip_horizontal:n { -2 \col@sep }
3745   }
3746 }

```

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

```

3747 \tl_const:Nn \c_@@_preamble_last_col_tl
3748 {
3749   >
3750   {
3751     \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).

```

3752     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

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

```

3753     \bool_gset_true:N \g_@@_last_col_found_bool
3754     \int_gincr:N \c@jCol
3755     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3756     \hbox_set:Nw \l_@@_cell_box
3757     \@@_math_toggle:
3758     \@@_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”.

```

3759     \int_compare:nNnT \c@iRow > \c_zero_int
3760     {
3761       \bool_lazy_or:nnT
3762       { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3763       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3764       {
3765         \l_@@_code_for_last_col_tl
3766         \xglobal \colorlet { nicematrix-last-col } { . }
3767       }
3768     }
3769   }
3770   l
3771   <

```

```

3772 {
3773   \@@_math_toggle:
3774   \hbox_set_end:
3775   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3776   \@@_adjust_size_box:
3777   \@@_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.

```

3778   \dim_gset:Nn \g_@@_width_last_col_dim
3779   { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3780   \skip_horizontal:n { -2 \col@sep }

```

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

```

3781   \hbox_overlap_right:n
3782   {
3783     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3784     {
3785       \skip_horizontal:N \l_@@_right_delim_dim
3786       \skip_horizontal:N \l_@@_right_margin_dim
3787       \skip_horizontal:N \l_@@_extra_right_margin_dim
3788       \@@_node_cell:
3789     }
3790   }
3791   \bool_gset_false:N \g_@@_empty_cell_bool
3792 }
3793 }

```

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

```

3794 \NewDocumentEnvironment { NiceArray } { }
3795 {
3796   \bool_gset_false:N \g_@@_delims_bool
3797   \str_if_empty:NT \g_@@_name_env_str
3798   { \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).

```

3799   \NiceArrayWithDelims . .
3800 }
3801 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

3802 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3803 {
3804   \NewDocumentEnvironment { #1 NiceArray } { }
3805   {
3806     \bool_gset_true:N \g_@@_delims_bool
3807     \str_if_empty:NT \g_@@_name_env_str
3808     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3809     \@@_test_if_math_mode:
3810     \NiceArrayWithDelims #2 #3
3811   }
3812   { \endNiceArrayWithDelims }
3813 }
3814 \@@_def_env:NNN p ( )
3815 \@@_def_env:NNN b [ ]
3816 \@@_def_env:NNN B \{ \}
3817 \@@_def_env:NNN v \vert \vert
3818 \@@_def_env:NNN V \Vert \Vert

```


13 The environment {NiceMatrix} and its variants

13.1 Definition of {pNiceMatrix}

```

3819 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3820 {
3821   \bool_set_false:N \l_@@_preamble_bool
3822   \tl_clear:N \l_tmpa_tl
3823   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3824     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3825   \tl_put_right:Nn \l_tmpa_tl
3826     {
3827       *
3828       {
3829         \int_case:nnF \l_@@_last_col_int
3830         {
3831           { -2 } { \c@MaxMatrixCols }
3832           { -1 } { \int_eval:n { \c@MaxMatrixCols + 1 } }

```

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

```

3833       }
3834       { \int_eval:n { \l_@@_last_col_int - 1 } }
3835     }
3836     { #2 }
3837   }
3838   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3839   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3840 }
3841 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3842 \clist_map_inline:nn { p , b , B , v , V }
3843 {
3844   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3845   {
3846     \bool_gset_true:N \g_@@_delims_bool
3847     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3848     \int_if_zero:nT \l_@@_last_col_int
3849     {
3850       \bool_set_true:N \l_@@_last_col_without_value_bool
3851       \int_set:Nn \l_@@_last_col_int { -1 }
3852     }
3853     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3854     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3855   }
3856   { \use:c { end #1 NiceArray } }
3857 }

```

We define also an environment {NiceMatrix}

```

3858 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3859 {
3860   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3861   \int_if_zero:nT \l_@@_last_col_int
3862   {
3863     \bool_set_true:N \l_@@_last_col_without_value_bool
3864     \int_set:Nn \l_@@_last_col_int { -1 }
3865   }
3866   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3867   \bool_lazy_or:nnT
3868     \l_@@_except_borders_bool
3869     { \clist_if_empty_p:N \l_@@_vlines_clist }
3870     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3871   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3872 }

```

```
3873 { \endNiceArray }
```

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

```
3874 \cs_new_protected:Npn \@@_NotEmpty:
3875 { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

13.2 The key `renew-matrix`

```
3876 \cs_set_protected:Npn \@@_renew_matrix:
3877 {
3878   \tl_map_inline:nn { pvVbB }
3879   { \RenewEnvironmentCopy { ##1matrix } { ##1NiceMatrix } }
3880 }
```

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

```
3881 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3882 {
```

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`.

```
3883   \dim_compare:nNtT\l_@@_width_dim = \c_zero_dim
3884   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3885   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3886   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3887   \tl_if_empty:NF \l_@@_short_caption_tl
3888   {
3889     \tl_if_empty:NT \l_@@_caption_tl
3890     {
3891       \@@_error_or_warning:n { short-caption~without~caption }
3892       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3893     }
3894   }
3895   \tl_if_empty:NF \l_@@_label_tl
3896   {
3897     \tl_if_empty:NT \l_@@_caption_tl
3898     { \@@_error_or_warning:n { label~without~caption } }
3899   }
3900   \NewDocumentEnvironment { TabularNote } { b }
3901   {
3902     \bool_if:NTF \l_@@_in_code_after_bool
3903     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3904     {
3905       \tl_if_empty:NF \g_@@_tabularnote_tl
3906       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3907       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3908     }
3909   }
3910   { }
3911   \@@_settings_for_tabular:
3912   \NiceArray { #2 }
3913 }
3914 { \endNiceArray }
3915 \cs_new_protected:Npn \@@_settings_for_tabular:
3916 {
3917   \bool_set_true:N \l_@@_tabular_bool
3918   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3919   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3920   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3921 }
```

```

3922 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3923 {
3924   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3925   \dim_set:Nn \l_@@_width_dim { #1 }
3926   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3927   \@@_settings_for_tabular:
3928   \NiceArray { #3 }
3929 }
3930 {
3931   \endNiceArray
3932   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3933   { \@@_error:n { NiceTabularX~without~X } }
3934 }

3935 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3936 {
3937   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3938   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3939   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3940   \@@_settings_for_tabular:
3941   \NiceArray { #3 }
3942 }
3943 { \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`).

```

3944 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3945 {
3946   \bool_lazy_all:nT
3947   {
3948     { \dim_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3949     { \l_@@_hvlines_bool }
3950     { ! \g_@@_delims_bool }
3951     { ! \l_@@_except_borders_bool }
3952   }
3953   {
3954     \bool_set_true:N \l_@@_except_borders_bool
3955     \clist_if_empty:NF \l_@@_corners_clist
3956     { \@@_error:n { hvlines,~rounded-corners-and~corners } }
3957     \tl_gput_right:Nn \g_@@_rules_tl
3958     {
3959       \@@_stroke_block:nnnnn
3960       {
3961         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3962         draw = \l_@@_rules_color_tl
3963       }
3964       { 1 } { 1 } { \int_use:N \c@iRow } { \int_use:N \c@jCol }
3965     }
3966   }
3967 }

3968 \cs_new_protected:Npn \@@_after_array:
3969 {

```

There was a `\hook_gput_code:nnn { env / tabular / begin } { nicematrix }` in the command `\@@_pre_array_after_CodeBefore:` in order to come back to the standard definition of

`\multicolumn` (in the tabulars used by the final user in the cells of our array of `nicematrix`) and maybe another linked to `colortbl`.

```
3970 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3971 \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.

```
3972 \bool_if:NT \g_@@_last_col_found_bool
3973 { \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`.

```
3974 \bool_if:NT \l_@@_last_col_without_value_bool
3975 { \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.

```
3976 \bool_if:NT \l_@@_last_row_without_value_bool
3977 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }
```

```
3978 \tl_gput_right:Ne \g_@@_aux_tl
3979 {
3980   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3981   {
3982     \int_use:N \l_@@_first_row_int ,
3983     \int_use:N \c@iRow ,
3984     \int_use:N \g_@@_row_total_int ,
3985     \int_use:N \l_@@_first_col_int ,
3986     \int_use:N \c@jCol ,
3987     \int_use:N \g_@@_col_total_int
3988   }
3989 }
3990 \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`).

```
3991 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3992 {
3993   \tl_gput_right:Ne \g_@@_aux_tl
3994   {
3995     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3996     { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }
3997   }
3998 }
3999 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
4000 {
4001   \tl_gput_right:Ne \g_@@_aux_tl
4002   {
4003     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
4004     { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
4005     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
4006     { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
4007   }
4008 }
```

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

```
4009 \@@_create_diag_nodes:
```

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

```

4010 \pgfpicture
4011 \@@_create_aliases_last:
4012 \str_if_empty:NF \l_@@_name_str \@@_create_alias_nodes:
4013 \endpgfpicture

```

By default, the diagonal lines will be parallelized¹². There are two types of diagonals lines: the `\Ddots` diagonals and the `\Iddots` diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current `{NiceArray}` environment.

```

4014 \bool_if:NT \l_@@_parallelize_diags_bool
4015 {
4016   \int_gzero:N \g_@@_ddots_int
4017   \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.

```

4018   \dim_gzero:N \g_@@_delta_x_one_dim
4019   \dim_gzero:N \g_@@_delta_y_one_dim
4020   \dim_gzero:N \g_@@_delta_x_two_dim
4021   \dim_gzero:N \g_@@_delta_y_two_dim
4022 }
4023 \bool_set_false:N \l_@@_initial_open_bool
4024 \bool_set_false:N \l_@@_final_open_bool

```

If the option `small` is used, the values `\l_@@_xdots_radius_dim` and `\l_@@_xdots_inter_dim` (used to draw the dotted lines created by `\hdottedline` and `\vdottedline` and also for all the other dotted lines when `line-style` is equal to `standard`, which is the initial value) are changed.

```

4025 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

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

```

4026 \@@_draw_dotted_lines:

```

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

```

4027 \clist_if_empty:NF \l_@@_corners_clist
4028 {
4029   \bool_if:NTF \l_@@_no_cell_nodes_bool
4030   { \@@_error:n { corners~with~no~cell~nodes } }
4031   \@@_compute_corners:
4032 }

```

By design, we have computed the corners before the adjonction of `\g_@@_future_pos_of_blocks_seq` is used by `\EmptyRow` and `\EmptyColumn` in the `\CodeBefore`.

```

4033 \seq_gconcat:NNN \g_@@_pos_of_blocks_seq
4034 \g_@@_pos_of_blocks_seq
4035 \g_@@_future_pos_of_blocks_seq
4036 \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-*}`).

```

4037 \@@_adjust_pos_of_blocks_seq:
4038 \@@_deal_with_rounded_corners:
4039 \legacy_if:nF { measuring@ } \@@_draw_rules:
4040 \tl_gclear:N \g_@@_rules_tl

```

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

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

```

4041 \IfPackageLoadedT { tikz }
4042 {
4043   \tikzset
4044   {
4045     every-picture / .style =
4046     {
4047       overlay ,
4048       remember-picture ,
4049       name-prefix = \@@_env: -
4050     }
4051   }
4052 }
4053 \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
4054 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
4055 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
4056 \cs_set_eq:NN \OverBrace \@@_OverBrace
4057 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
4058 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
4059 \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.

```

4060 \legacy_if:nF { measuring@ } \g_@@_pre_code_after_tl
4061 \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.

```

4062 \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).

```

4063 \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`).

```

4064 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
4065 { \@@_rescan_for_spanish:N \g_nicematrix_code_after_tl }

```

And here's the `\CodeAfter`. Since the `\CodeAfter` may begin with an “argument” between square brackets of the options, we extract and treat that potential “argument” with the command `\@@_CodeAfter_keys:`.

```

4066 \bool_set_true:N \l_@@_in_code_after_bool
4067 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4068 \scan_stop:
4069 \tl_gclear:N \g_nicematrix_code_after_tl
4070 \clist_if_empty:NF \g_@@_col_with_trees_clist \@@_draw_trees:
4071 \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.

```

4072 \seq_if_empty:NF \g_@@_rowlistcolors_seq \@@_clear_rowlistcolors_seq:
4073 \tl_if_empty:NF \g_@@_pre_code_before_tl
4074 {
4075   \tl_gput_right:Ne \g_@@_aux_tl
4076   {
4077     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
4078     { \exp_not:o \g_@@_pre_code_before_tl }
4079   }
4080   \tl_gclear:N \g_@@_pre_code_before_tl

```

```

4081     }
4082     \tl_if_empty:NF \g_nicematrix_code_before_tl
4083     {
4084         \tl_gput_right:Ne \g_@@_aux_tl
4085         {
4086             \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
4087             { \exp_not:o \g_nicematrix_code_before_tl }
4088         }
4089         \tl_gclear:N \g_nicematrix_code_before_tl
4090     }

4091     \str_gclear:N \g_@@_name_env_str
4092     \@@_restore_iRow_jCol:

```

The command `\CT@arc@` contains the instruction of color for the rules of the array¹³. This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able to use color for the rules of the array when `colortbl` is *not* loaded. That's why we do the following instruction which is in the patch of the end of arrays done by `colortbl`.

```

4093     \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
4094 }

4095 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
4096 {
4097     \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
4098     \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }

4099     \dim_set:Nn \l_@@_xdots_shorten_start_dim
4100     { 0.6 \l_@@_xdots_shorten_start_dim }
4101     \dim_set:Nn \l_@@_xdots_shorten_end_dim
4102     { 0.6 \l_@@_xdots_shorten_end_dim }
4103 }

```

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

The following command will extract the potential options (between square brackets) at the beginning of the `\CodeAfter` (that is to say, when `\CodeAfter` is used, the options of that “command” `\CodeAfter`). Idem for the `\CodeBefore`.

```

4104 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4105 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

4106 \cs_new_protected:Npn \@@_create_alias_nodes:
4107 {
4108     \int_step_inline:nn \c@iRow
4109     {
4110         \pgfnodealias
4111         { \l_@@_name_str - ##1 - last }
4112         { \@@_env: - ##1 - \int_use:N \c@jCol }
4113     }
4114     \int_step_inline:nn \c@jCol
4115     {
4116         \pgfnodealias
4117         { \l_@@_name_str - last - ##1 }
4118         { \@@_env: - \int_use:N \c@iRow - ##1 }
4119     }
4120     \pgfnodealias
4121     { \l_@@_name_str - last - last }
4122     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
4123 }

```

¹³e.g. `\color[rgb]{0.5,0.5,0}`

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).

```

4124 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4125 {
4126   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4127   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4128 }

```

The following command must *not* be protected.

```

4129 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4130 {
4131   { #1 }
4132   { #2 }
4133   {
4134     \int_compare:nNnTF { #3 } > { 98 }
4135     { \int_use:N \c@iRow }
4136     { #3 }
4137   }
4138   {
4139     \int_compare:nNnTF { #4 } > { 98 }
4140     { \int_use:N \c@jCol }
4141     { #4 }
4142   }
4143   { #5 }
4144 }

```

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).

```

4145 \AtBeginDocument
4146 {
4147   \cs_new_protected:Npe \@@_draw_dotted_lines:
4148   {
4149     \c_@@_pgfortikzpicture_tl
4150     \@@_draw_dotted_lines_i:
4151     \c_@@_endpgfortikzpicture_tl
4152   }
4153 }

```

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

```

4154 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4155 {
4156   \pgfrememberpicturepositiononpagetrue
4157   \pgf@relevantforpicturesizefalse
4158   \g_@@_HVdotsfor_lines_tl
4159   \g_@@_Vdots_lines_tl
4160   \g_@@_Ddots_lines_tl
4161   \g_@@_Idots_lines_tl
4162   \g_@@_Cdots_lines_tl
4163   \g_@@_Ldots_lines_tl
4164 }

4165 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4166 {
4167   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4168   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4169 }

```


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

```

4170 \pgfdeclareshape { @@_diag_node }
4171 {
4172   \savedanchor { \five }
4173   {
4174     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4175     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4176   }
4177   \anchor { 5 } { \five }
4178   \anchor { center } { \pgfpointorigin }
4179   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4180   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4181   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4182   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4183   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4184   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4185   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4186   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4187   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4188   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4189 }

```

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).

```

4190 \cs_new_protected:Npn \@@_create_diag_nodes:
4191 {
4192   \pgfpicture
4193   \pgfrememberpicturepositiononpagetrue
4194   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4195   {
4196     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4197     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4198     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4199     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4200     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4201     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4202     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4203     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4204     \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.

```

4205     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4206     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4207     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4208     \str_if_empty:NF \l_@@_name_str
4209     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4210   }

```

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

```

4211     \int_set:Nn \l_tmpa_int { 1 + \int_max:nn \c@iRow \c@jCol } % modified
4212     \@@_qpoint:n { row - \int_min:nn \l_tmpa_int { \c@iRow + 1 } }
4213     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4214     \@@_qpoint:n { col - \int_min:nn \l_tmpa_int { \c@jCol + 1 } }
4215     \pgfcoordinate
4216     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4217     \pgfnodealias
4218     { \@@_env: - last }
4219     { \@@_env: - \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
4220     \str_if_empty:NF \l_@@_name_str
4221     {

```

```

4222     \pgfnodealias
4223     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4224     { \@@_env: - \int_use:N \l_tmpa_int }
4225     \pgfnodealias
4226     { \l_@@_name_str - last }
4227     { \@@_env: - last }
4228   }
4229   \endpgfpicture
4230 }

```

16 We draw the dotted lines

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

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

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.

```

4231 \cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
4232 {

```

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

```

4233 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4234 \l_@@_initial_i_int = #1
4235 \l_@@_initial_j_int = #2
4236 \l_@@_final_i_int = #1
4237 \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.

```

4238 \let \l_@@_stop_loop_bool \c_false_bool
4239 \bool_do_until:Nn \l_@@_stop_loop_bool
4240 {

```

We test if we are still in the matrix.

```

4241 \advance \l_@@_final_i_int by #3
4242 \advance \l_@@_final_j_int by #4
4243 \let \l_@@_final_open_bool \c_false_bool
4244 \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4245 \if_int_compare:w #3 = \c_one_int
4246 \let \l_@@_final_open_bool \c_true_bool
4247 \else:
4248 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4249 \let \l_@@_final_open_bool \c_true_bool
4250 \fi:
4251 \fi:
4252 \else:
4253 \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4254 \if_int_compare:w #4 = -1
4255 \let \l_@@_final_open_bool \c_true_bool
4256 \fi:
4257 \else:
4258 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4259 \if_int_compare:w #4 = \c_one_int
4260 \let \l_@@_final_open_bool \c_true_bool
4261 \fi:
4262 \fi:
4263 \fi:
4264 \fi:
4265 \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.

```

4266 {

```

We do a step backwards.

```

4267 \advance \l_@@_final_i_int by - #3
4268 \advance \l_@@_final_j_int by - #4
4269 \let \l_@@_stop_loop_bool \c_true_bool
4270 }

```

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`.

```

4271     {
4272         \cs_if_exist:cTF
4273         {
4274             @@ _ dotted _
4275             \int_use:N \l_@@_final_i_int -
4276             \int_use:N \l_@@_final_j_int
4277         }
4278         {
4279             \advance \l_@@_final_i_int by - #3
4280             \advance \l_@@_final_j_int by - #4
4281             \let \l_@@_final_open_bool \c_true_bool
4282             \let \l_@@_stop_loop_bool \c_true_bool
4283         }
4284     {
4285         \cs_if_exist:cTF
4286         {
4287             pgf @ sh @ ns @ \@@_env:
4288             - \int_use:N \l_@@_final_i_int
4289             - \int_use:N \l_@@_final_j_int
4290         }
4291         { \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.

```

4292     {
4293         \cs_set_nopar:cpn
4294         {
4295             @@ _ dotted _
4296             \int_use:N \l_@@_final_i_int -
4297             \int_use:N \l_@@_final_j_int
4298         }
4299         { }
4300     }
4301 }
4302 }
4303 }

```

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

```

4304     \let \l_@@_stop_loop_bool \c_false_bool

```

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

```

4305     \l_tmpa_int = \l_@@_col_min_int
4306     \advance \l_tmpa_int by -1
4307     \bool_do_until:Nn \l_@@_stop_loop_bool
4308     {
4309         \advance \l_@@_initial_i_int by - #3
4310         \advance \l_@@_initial_j_int by - #4
4311         \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.

```

4312     \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4313     \if_int_compare:w #3 = \c_one_int
4314         \let \l_@@_initial_open_bool \c_true_bool
4315     \else:

```

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

```

4316         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4317         \let \l_@@_initial_open_bool \c_true_bool
4318     \fi:
4319 \fi:
4320 \else:
4321     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4322     \if_int_compare:w #4 = \c_one_int
4323     \let \l_@@_initial_open_bool \c_true_bool
4324     \fi:
4325 \else:
4326     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4327     \if_int_compare:w #4 = -1
4328     \let \l_@@_initial_open_bool \c_true_bool
4329     \fi:
4330 \fi:
4331 \fi:
4332 \fi:
4333 \bool_if:NTF \l_@@_initial_open_bool
4334 {
4335     \advance \l_@@_initial_i_int by #3
4336     \advance \l_@@_initial_j_int by #4
4337     \let \l_@@_stop_loop_bool \c_true_bool
4338 }
4339 {
4340     \cs_if_exist:cTF
4341     {
4342         @@ _ dotted _
4343         \int_use:N \l_@@_initial_i_int -
4344         \int_use:N \l_@@_initial_j_int
4345     }
4346     {
4347         \advance \l_@@_initial_i_int by #3
4348         \advance \l_@@_initial_j_int by #4
4349         \let \l_@@_initial_open_bool \c_true_bool
4350         \let \l_@@_stop_loop_bool \c_true_bool
4351     }
4352     {
4353         \cs_if_exist:cTF
4354         {
4355             pgf @ sh @ ns @ \@@_env:
4356             - \int_use:N \l_@@_initial_i_int
4357             - \int_use:N \l_@@_initial_j_int
4358         }
4359         { \let \l_@@_stop_loop_bool \c_true_bool }
4360         {
4361             \cs_set_nopar:cpn
4362             {
4363                 @@ _ dotted _
4364                 \int_use:N \l_@@_initial_i_int -
4365                 \int_use:N \l_@@_initial_j_int
4366             }
4367             { }
4368         }
4369     }
4370 }
4371 }
```

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.

```

4372 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4373 {
4374     { \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`.

```

4375         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4376         { \int_use:N \l_@@_final_i_int }
4377         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4378         { }
4379     }
4380 }

```

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.

```

4381 \cs_new_protected:Npn \@@_open_shorten:
4382 {
4383     \bool_if:NT \l_@@_initial_open_bool
4384     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4385     \bool_if:NT \l_@@_final_open_bool
4386     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4387 }

```

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).

```

4388 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4389 {
4390     \int_set_eq:NN \l_@@_row_min_int \c_one_int
4391     \int_set_eq:NN \l_@@_col_min_int \c_one_int
4392     \int_set_eq:NN \l_@@_row_max_int \c_iRow
4393     \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.

```

4394     \seq_if_empty:NTF \g_@@_submatrix_seq
4395     { \cs_set_eq:NN \@@_adjust_to_submatrix:nn \use_none:nn }
4396     {

```

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`.

```

4397         \seq_map_inline:Nn \g_@@_submatrix_seq
4398         { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4399     }
4400 }

```

#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.

```

4401 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4402 {
4403   \if_int_compare:w #3 > #1
4404   \else:
4405     \if_int_compare:w #1 > #5
4406     \else:
4407       \if_int_compare:w #4 > #2
4408       \else:
4409         \if_int_compare:w #2 > #6
4410         \else:
4411           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4412           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4413           \if_int_compare:w \l_@@_row_max_int > #5 \l_@@_row_max_int = #5 \fi:
4414           \if_int_compare:w \l_@@_col_max_int > #6 \l_@@_col_max_int = #6 \fi:
4415         \fi:
4416       \fi:
4417     \fi:
4418   \fi:
4419 }

4420 \cs_new_protected:Npn \@@_set_initial_coords:
4421 {
4422   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4423   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4424 }
4425 \cs_new_protected:Npn \@@_set_final_coords:
4426 {
4427   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4428   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4429 }
4430 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4431 {
4432   \pgfpointanchor
4433   {
4434     \@@_env:
4435     - \int_use:N \l_@@_initial_i_int
4436     - \int_use:N \l_@@_initial_j_int
4437   }
4438   { #1 }
4439   \@@_set_initial_coords:
4440 }
4441 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4442 {
4443   \pgfpointanchor
4444   {
4445     \@@_env:
4446     - \int_use:N \l_@@_final_i_int
4447     - \int_use:N \l_@@_final_j_int
4448   }
4449   { #1 }
4450   \@@_set_final_coords:
4451 }
4452 \cs_new_protected:Npn \@@_open_x_initial_dim:
4453 {
4454   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4455   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4456   {
4457     \cs_if_exist:cT

```

```

4458     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4459     {
4460       \pgfpointanchor
4461       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4462       { west }
4463       \dim_set:Nn \l_@@_x_initial_dim
4464       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4465     }
4466   }

```

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

```

4467   \dim_compare:nNtT \l_@@_x_initial_dim = \c_max_dim
4468   {
4469     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4470     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4471     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4472   }
4473 }

4474 \cs_new_protected:Npn \@@_open_x_final_dim:
4475 {
4476   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4477   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4478   {
4479     \cs_if_exist:cT
4480     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4481     {
4482       \pgfpointanchor
4483       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4484       { east }
4485       \dim_compare:nNtT \pgf@x > \l_@@_x_final_dim
4486       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4487     }
4488   }

```

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

```

4489   \dim_compare:nNtT \l_@@_x_final_dim = { - \c_max_dim }
4490   {
4491     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4492     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4493     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4494   }
4495 }

```

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.

```

4496 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4497 {
4498   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4499   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4500   {
4501     \@@_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.

```

4502   \bool_if:NT \g_@@_aux_found_bool
4503   {
4504     {
4505       \@@_open_shorten:
4506       \int_if_zero:nTF { #1 }
4507       { \color { nicematrix-first-row } }
4508     }

```

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.

```

4509         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4510         { \color { nicematrix-last-row } }
4511     }
4512     \keys_set:nn { nicematrix / xdots } { #3 }
4513     \@@_color:o \l_@@_xdots_color_tl
4514     \@@_actually_draw_Ldots:
4515 }
4516 }
4517 }
4518 }
```

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`.

```

4519 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4520 {
4521     \bool_if:NTF \l_@@_initial_open_bool
4522     {
4523         \@@_open_x_initial_dim:
4524         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4525         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4526     }
4527     { \@@_set_initial_coords_from_anchor:n { base~east } }
4528     \bool_if:NTF \l_@@_final_open_bool
4529     {
4530         \@@_open_x_final_dim:
4531         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4532         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4533     }
4534     { \@@_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.

```

4535     \bool_lazy_all:nTF
4536     {
4537         \l_@@_initial_open_bool
4538         \l_@@_final_open_bool
4539         { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4540     }
4541     {
4542         \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4543         \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4544     }
```

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 (?).

```

4545     {
4546         \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4547         \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4548     }
4549     \@@_draw_line:
4550 }
```

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.

```

4551 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4552 {
4553   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4554   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4555   {
4556     \@@_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.

```

4557     \bool_if:NT \g_@@_aux_found_bool
4558     {
4559       {
4560         \@@_open_shorten:
4561         \int_if_zero:nTF { #1 }
4562         { \color { nicematrix-first-row } }
4563         {

```

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.

```

4564         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4565         { \color { nicematrix-last-row } }
4566       }
4567       \keys_set:nn { nicematrix / xdots } { #3 }
4568       \@@_color:o \l_@@_xdots_color_tl
4569       \@@_actually_draw_Cdots:
4570     }
4571   }
4572 }
4573 }

```

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`.

```

4574 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4575 {
4576   \bool_if:NTF \l_@@_initial_open_bool
4577   \@@_open_x_initial_dim:
4578   { \@@_set_initial_coords_from_anchor:n { mid-east } }
4579   \bool_if:NTF \l_@@_final_open_bool
4580   \@@_open_x_final_dim:
4581   { \@@_set_final_coords_from_anchor:n { mid-west } }
4582   \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4583   {
4584     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4585     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4586     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4587     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4588     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4589   }
4590   {
4591     \bool_if:NT \l_@@_initial_open_bool
4592     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4593     \bool_if:NT \l_@@_final_open_bool

```

```

4594         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4595     }
4596     \@@_draw_line:
4597 }
4598 \cs_new_protected:Npn \@@_open_y_initial_dim:
4599 {
4600     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4601     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4602     {
4603         \cs_if_exist:cT
4604         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4605         {
4606             \pgfpointanchor
4607             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4608             { north }
4609             \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4610             { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4611         }
4612     }
4613     \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4614     {
4615         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4616         \dim_set:Nn \l_@@_y_initial_dim
4617         {
4618             \fp_to_dim:n
4619             {
4620                 \pgf@y
4621                 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4622             }
4623         }
4624     }
4625 }
4626 \cs_new_protected:Npn \@@_open_y_final_dim:
4627 {
4628     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4629     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4630     {
4631         \cs_if_exist:cT
4632         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4633         {
4634             \pgfpointanchor
4635             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4636             { south }
4637             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4638             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4639         }
4640     }
4641     \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4642     {
4643         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4644         \dim_set:Nn \l_@@_y_final_dim
4645         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4646     }
4647 }

```

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.

```

4648 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4649 {
4650     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4651     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4652     {
4653         \@@_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.

```

4654     \bool_if:NT \g_@@_aux_found_bool
4655     {
4656     {
4657         \@@_open_shorten:
4658         \int_if_zero:nTF { #2 }
4659         { \color { nicematrix-first-col } }
4660         {
4661             \int_compare:nNnT { #2 } = \l_@@_last_col_int
4662             { \color { nicematrix-last-col } }
4663         }
4664         \keys_set:nn { nicematrix / xdots } { #3 }
4665         \@@_color:o \l_@@_xdots_color_tl
4666         \bool_if:NTF \l_@@_Vbrace_bool
4667         \@@_actually_draw_Vbrace:
4668         \@@_actually_draw_Vdots:
4669     }
4670     }
4671 }
4672 }
```

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`.

```

4673 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4674 {
4675     \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4676     \@@_actually_draw_Vdots_i:
4677     \@@_actually_draw_Vdots_ii:
4678     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4679     \@@_draw_line:
4680 }
```

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

```

4681 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4682 {
4683     \@@_open_y_initial_dim:
4684     \@@_open_y_final_dim:
4685     \int_if_zero:nTF \l_@@_initial_j_int
```

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

```

4686     {
4687         \@@_qpoint:n { col - 1 }
4688         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4689         \dim_sub:Nn \l_@@_x_initial_dim
4690         { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4691     }
4692     {
4693         \bool_lazy_and:nnTF
4694         { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4695         { \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”.

```

4696     {
4697         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4698         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4699         \dim_add:Nn \l_@@_x_initial_dim
4700             { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4701     }

```

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

```

4702     {
4703         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4704         \dim_set_eq:NN \l_tmpa_dim \pgf@x
4705         \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4706         \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4707     }
4708 }
4709 }

```

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.

```

4710 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4711 {
4712     \bool_set_false:N \l_tmpa_bool
4713     \bool_if:NF \l_@@_initial_open_bool
4714     {
4715         \bool_if:NF \l_@@_final_open_bool
4716         {
4717             \@@_set_initial_coords_from_anchor:n { south-west }
4718             \@@_set_final_coords_from_anchor:n { north-west }
4719             \bool_set:Nn \l_tmpa_bool
4720                 { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4721         }
4722     }

```

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

```

4723     \bool_if:NTF \l_@@_initial_open_bool
4724     {
4725         \@@_open_y_initial_dim:
4726         \@@_set_final_coords_from_anchor:n { north }
4727         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4728     }
4729     {
4730         \@@_set_initial_coords_from_anchor:n { south }
4731         \bool_if:NTF \l_@@_final_open_bool
4732         \@@_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.

```

4733     {
4734         \@@_set_final_coords_from_anchor:n { north }
4735         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4736         {
4737             \dim_set:Nn \l_@@_x_initial_dim
4738             {
4739                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4740                     \l_@@_x_initial_dim \l_@@_x_final_dim
4741             }
4742         }
4743     }
4744 }
4745 }

```

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.`

```

4746 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4747 {
4748   \bool_if:NTF \l_@@_initial_open_bool
4749     \@@_open_y_initial_dim:
4750     { \@@_set_initial_coords_from_anchor:n { south } }
4751   \bool_if:NTF \l_@@_final_open_bool
4752     \@@_open_y_final_dim:
4753     { \@@_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.

```

4754   \int_if_zero:nTF \l_@@_initial_j_int
4755   {
4756     \@@_qpoint:n { col - 1 }
4757     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4758     \dim_sub:Nn \l_@@_x_initial_dim
4759     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4760   }

```

Elsewhere, the brace must be drawn left flush.

```

4761   {
4762     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4763     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4764     \dim_add:Nn \l_@@_x_initial_dim
4765     { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4766   }

```

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

```

4767   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4768   \@@_draw_line:
4769 }

```

```

4770 \cs_new:Npn \@@_colsep:
4771 { \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.

```

4772 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4773 {
4774   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4775   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4776   {
4777     \@@_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.

```

4778     \bool_if:NT \g_@@_aux_found_bool
4779     {
4780     {
4781         \@@_open_shorten:
4782         \keys_set:nn { nicematrix / xdots } { #3 }
4783         \@@_color:o \l_@@_xdots_color_tl
4784         \@@_actually_draw_Ddots:
4785     }
4786     }
4787 }
4788 }

```

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`.

```

4789 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4790 {
4791     \bool_if:NTF \l_@@_initial_open_bool
4792     {
4793         \@@_open_y_initial_dim:
4794         \@@_open_x_initial_dim:
4795     }
4796     { \@@_set_initial_coords_from_anchor:n { south~east } }
4797     \bool_if:NTF \l_@@_final_open_bool
4798     {
4799         \@@_open_x_final_dim:
4800         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4801     }
4802     { \@@_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.

```

4803     \bool_if:NT \l_@@_parallelize_diags_bool
4804     {
4805         \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).

```

4806         \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.

```

4807     {
4808         \dim_gset:Nn \g_@@_delta_x_one_dim
4809         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4810         \dim_gset:Nn \g_@@_delta_y_one_dim
4811         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4812     }

```

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`.

```

4813     {
4814         \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4815         {
4816             \dim_set:Nn \l_@@_y_final_dim
4817             {
4818                 \l_@@_y_initial_dim +
4819                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4820                 \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4821             }
4822         }
4823     }
4824 }
4825 \@@_draw_line:
4826 }

```

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.

```

4827 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4828 {
4829     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4830     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4831     {
4832         \@@_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.

```

4833         \bool_if:NT \g_@@_aux_found_bool
4834         {
4835             {
4836                 \@@_open_shorten:
4837                 \keys_set:nn { nicematrix / xdots } { #3 }
4838                 \@@_color:o \l_@@_xdots_color_tl
4839                 \@@_actually_draw_Iddots:
4840             }
4841         }
4842     }
4843 }

```

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`.

```

4844 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4845 {
4846     \bool_if:NTF \l_@@_initial_open_bool
4847     {
4848         \@@_open_y_initial_dim:
4849         \@@_open_x_initial_dim:
4850     }
4851     { \@@_set_initial_coords_from_anchor:n { south-west } }
4852     \bool_if:NTF \l_@@_final_open_bool

```

```

4853 {
4854   \l_@@_open_y_final_dim:
4855   \l_@@_open_x_final_dim:
4856 }
4857 { \l_@@_set_final_coords_from_anchor:n { north-east } }
4858 \bool_if:NT \l_@@_parallelize_diags_bool
4859 {
4860   \int_gincr:N \g_@@_iddots_int
4861   \int_compare:nNnTF \g_@@_iddots_int = 1
4862   {
4863     \dim_gset:Nn \g_@@_delta_x_two_dim
4864     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4865     \dim_gset:Nn \g_@@_delta_y_two_dim
4866     { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4867   }
4868   {
4869     \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4870     {
4871       \dim_set:Nn \l_@@_y_final_dim
4872       {
4873         \l_@@_y_initial_dim +
4874         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4875         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4876       }
4877     }
4878   }
4879 }
4880 \l_@@_draw_line:
4881 }

```

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`

```

4882 \cs_new_protected:Npn \l_@@_draw_line:
4883 {
4884   \pgfrememberpicturepositiononpagetrue
4885   \pgf@relevantforpicturesizefalse
4886   \bool_lazy_or:nnTF
4887   \l_@@_dotted_bool
4888   { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4889   \l_@@_draw_standard_dotted_line:
4890   \l_@@_draw_unstandard_dotted_line:
4891 }

```

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.

```

4892 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4893 {
4894   \begin { scope }
4895     \@@_draw_unstandard_dotted_line:o
4896     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4897 }

```

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.

```

4898 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4899 {
4900   \@@_draw_unstandard_dotted_line:nooo
4901   { #1 }
4902   \l_@@_xdots_up_tl
4903   \l_@@_xdots_down_tl
4904   \l_@@_xdots_middle_tl
4905 }
4906 \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.

```

4907 \AtBeginDocument
4908 {
4909   \IfPackageLoadedT { tikz }
4910   {
4911     \tikzset
4912     {
4913       @@_node_above / .style = { sloped , above } ,
4914       @@_node_below / .style = { sloped , below } ,
4915       @@_node_middle / .style =
4916       {
4917         sloped ,
4918         inner~sep = \c_@@_innersep_middle_dim
4919       }
4920     }
4921   }
4922 }

4923 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4924 {

```

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.

```

4925 \dim_zero_new:N \l_@@_l_dim
4926 \dim_set:Nn \l_@@_l_dim
4927 {
4928   \fp_to_dim:n
4929   {
4930     sqrt
4931     (
4932       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4933       +
4934       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4935     )
4936   }
4937 }

```

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.

```

4938   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4939   {
4940     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4941     \@@_draw_unstandard_dotted_line_i:
4942   }

```

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

```

4943   \bool_if:NT \l_@@_xdots_h_labels_bool
4944   {
4945     \tikzset
4946     {
4947       @@_node_above / .style = { auto = left } ,
4948       @@_node_below / .style = { auto = right } ,
4949       @@_node_middle / .style = { innersep = \c_@@_innersep_middle_dim }
4950     }
4951   }
4952   \tl_if_empty:nF { #4 }
4953   { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4954   \dim_zero:N \l_tmpa_dim
4955   \dim_zero:N \l_tmpb_dim
4956   \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
4957   {

```

We test whether the brace is vertical or horizontal.

```

4958     \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4959     { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
4960     { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
4961   }
4962   {
4963     \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
4964     {
4965       \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4966       { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
4967       { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
4968     }
4969   }
4970   \use:e
4971   {
4972     \exp_not:N \begin { scope }
4973     [ shift = {(\dim_use:N \l_tmpa_dim, \dim_use:N \l_tmpb_dim)} ]
4974   }
4975   \draw
4976   [ #1 ]
4977   ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
4978   -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4979   node [ @@_node_below ] { $ \scriptstyle #3 $ }
4980   node [ @@_node_above ] { $ \scriptstyle #2 $ }
4981   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4982   \end { scope }
4983   \end { scope }
4984 }
4985 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4986 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4987 {
4988   \dim_set:Nn \l_tmpa_dim
4989   {
4990     \l_@@_x_initial_dim
4991     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4992     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim

```

```

4993     }
4994     \dim_set:Nn \l_tmpb_dim
4995     {
4996         \l_@@_y_initial_dim
4997         + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4998         * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4999     }
5000     \dim_set:Nn \l_@@_tmpc_dim
5001     {
5002         \l_@@_x_final_dim
5003         - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5004         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5005     }
5006     \dim_set:Nn \l_@@_tmpd_dim
5007     {
5008         \l_@@_y_final_dim
5009         - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5010         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5011     }
5012     \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
5013     \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
5014     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
5015     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
5016 }

```

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

```

5017 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
5018 {
5019 {

```

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.

```

5020     \dim_zero_new:N \l_@@_l_dim
5021     \dim_set:Nn \l_@@_l_dim
5022     {
5023         \fp_to_dim:n
5024         {
5025             sqrt
5026             (
5027                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5028                 +
5029                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5030             )
5031         }
5032     }

```

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.

```

5033     \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
5034     {
5035         \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
5036         \@@_draw_standard_dotted_line_i:
5037     }
5038 }
5039 \bool_lazy_all:nF
5040 {
5041     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
5042     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
5043     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
5044 }

```

```

5045 \@@_labels_standard_dotted_line:
5046 }
5047 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
5048 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
5049 {

```

The number of dots will be $\l_1\text{tmpa_int} + 1$.

```

5050 \int_set:Nn \l_1tmpa_int
5051 {
5052 \dim_ratio:nn
5053 {
5054 \l_@@_l_dim
5055 - \l_@@_xdots_shorten_start_dim
5056 - \l_@@_xdots_shorten_end_dim
5057 }
5058 \l_@@_xdots_inter_dim
5059 }

```

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.

```

5060 \dim_set:Nn \l_1tmpa_dim
5061 {
5062 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5063 \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5064 }
5065 \dim_set:Nn \l_1tmpb_dim
5066 {
5067 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5068 \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5069 }

```

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.

```

5070 \dim_gadd:Nn \l_@@_x_initial_dim
5071 {
5072 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5073 \dim_ratio:nn
5074 {
5075 \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
5076 + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5077 }
5078 { 2 \l_@@_l_dim }
5079 }
5080 \dim_gadd:Nn \l_@@_y_initial_dim
5081 {
5082 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5083 \dim_ratio:nn
5084 {
5085 \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
5086 + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5087 }
5088 { 2 \l_@@_l_dim }
5089 }
5090 \pgf@relevantforpicturesizefalse
5091 \int_step_inline:nnn \c_zero_int \l_1tmpa_int
5092 {
5093 \pgfpathcircle
5094 { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5095 \l_@@_xdots_radius_dim
5096 \dim_add:Nn \l_@@_x_initial_dim \l_1tmpa_dim
5097 \dim_add:Nn \l_@@_y_initial_dim \l_1tmpb_dim
5098 }
5099 \pgfusepathqfill
5100 }

```

```

5101 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
5102 {
5103   \pgfscope
5104   \pgftransformshift
5105   {
5106     \pgfpointlineattime { 0.5 }
5107     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5108     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
5109   }
5110   \fp_set:Nn \l_tmpa_fp
5111   {
5112     atand
5113     (
5114       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
5115       \l_@@_x_final_dim - \l_@@_x_initial_dim
5116     )
5117   }
5118   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
5119   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
5120   \tl_if_empty:NF \l_@@_xdots_middle_tl
5121   {
5122     \begin { pgfscope }
5123     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
5124     \pgfnode
5125     { rectangle }
5126     { center }
5127     {
5128       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5129       {
5130         $ % $
5131         \scriptstyle \l_@@_xdots_middle_tl
5132         $ % $
5133       }
5134     }
5135     { }
5136     {
5137       \pgfsetfillcolor { white }
5138       \pgfusepath { fill }
5139     }
5140     \end { pgfscope }
5141   }
5142   \tl_if_empty:NF \l_@@_xdots_up_tl
5143   {
5144     \pgfnode
5145     { rectangle }
5146     { south }
5147     {
5148       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5149       {
5150         $ % $
5151         \scriptstyle \l_@@_xdots_up_tl
5152         $ % $
5153       }
5154     }
5155     { }
5156     { \pgfusepath { } }
5157   }
5158   \tl_if_empty:NF \l_@@_xdots_down_tl
5159   {
5160     \pgfnode
5161     { rectangle }
5162     { north }
5163     {

```



```

5164         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5165         {
5166             $ % $
5167             \scriptstyle \l_@@_xdots_down_tl
5168             $ % $
5169         }
5170     }
5171     { }
5172     { \pgfusepath { } }
5173 }
5174 \endpgfscope
5175 }

```

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.

```

5176 \AtBeginDocument
5177 {

```

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

```

5178     \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
5179     \cs_new_protected:Npn \@@_Ldots: { \@@_collect_options:n { \@@_Ldots_i } }
5180     \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5181     {
5182         \int_if_zero:nTF \c@jCol
5183         { \@@_error:nn { in~first~col } { \Ldots } }
5184         {
5185             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5186             { \@@_error:nn { in~last~col } { \Ldots } }
5187             {
5188                 \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5189                 { #1 , down = #2 , up = #3 , middle = #4 }
5190             }
5191         }
5192         \bool_if:NF \l_@@_nullify_dots_bool
5193         { \phantom { \ensuremath { \@@_old_ldots: } } }
5194         \bool_gset_true:N \g_@@_empty_cell_bool
5195     }

5196     \cs_new_protected:Npn \@@_Cdots: { \@@_collect_options:n { \@@_Cdots_i } }
5197     \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5198     {
5199         \int_if_zero:nTF \c@jCol
5200         { \@@_error:nn { in~first~col } { \Cdots } }
5201         {
5202             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5203             { \@@_error:nn { in~last~col } { \Cdots } }
5204             {
5205                 \@@_instruction_of_type:nnn { \c_false_bool } { \Cdots }

```

```

5206         { #1 , down = #2 , up = #3 , middle = #4 }
5207     }
5208 }
5209 \bool_if:NF \l_@@_nullify_dots_bool
5210 { \phantom { \ensuremath { \@@_old_cdots: } } }
5211 \bool_gset_true:N \g_@@_empty_cell_bool
5212 }

5213 \cs_new_protected:Npn \@@_Vdots: { \@@_collect_options:n { \@@_Vdots_i } }
5214 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5215 {
5216     \int_if_zero:nTF \c@iRow
5217     { \@@_error:nn { in~first~row } { \Vdots } }
5218     {
5219         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5220         { \@@_error:nn { in~last~row } { \Vdots } }
5221         {
5222             \@@_instruction_of_type:nnn { \c_false_bool } { \Vdots }
5223             { #1 , down = #2 , up = #3 , middle = #4 }
5224         }
5225     }
5226     \bool_if:NF \l_@@_nullify_dots_bool
5227     { \phantom { \ensuremath { \@@_old_vdots: } } }
5228     \bool_gset_true:N \g_@@_empty_cell_bool
5229 }

5230 \cs_new_protected:Npn \@@_Ddots: { \@@_collect_options:n { \@@_Ddots_i } }
5231 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5232 {
5233     \int_case:nnF \c@iRow
5234     {
5235         0 { \@@_error:nn { in~first~row } { \Ddots } }
5236         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5237     }
5238     {
5239         \int_case:nnF \c@jCol
5240         {
5241             0 { \@@_error:nn { in~first~col } { \Ddots } }
5242             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5243         }
5244         {
5245             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5246             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
5247             { #1 , down = #2 , up = #3 , middle = #4 }
5248         }
5249     }
5250 }
5251 \bool_if:NF \l_@@_nullify_dots_bool
5252 { \phantom { \ensuremath { \@@_old_ddots: } } }
5253 \bool_gset_true:N \g_@@_empty_cell_bool
5254 }

5255 \cs_new_protected:Npn \@@_Iddots:
5256 { \@@_collect_options:n { \@@_Iddots_i } }
5257 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5258 {
5259     \int_case:nnF \c@iRow
5260     {
5261         0 { \@@_error:nn { in~first~row } { \Iddots } }
5262         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5263     }

```

```

5264     {
5265         \int_case:nnF \c@jCol
5266         {
5267             0 { \@@_error:nn { in~first~col } { \Iddots } }
5268             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5269         }
5270         {
5271             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5272             \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { Iddots }
5273             { #1 , down = #2 , up = #3 , middle = #4 }
5274         }
5275     }
5276     \bool_if:NF \l_@@_nullify_dots_bool
5277     { \phantom { \ensuremath { \@@_old_iddots: } } }
5278     \bool_gset_true:N \g_@@_empty_cell_bool
5279 }
5280 }

```

End of the \AddToHook.

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

```

5281 \keys_define:nn { nicematrix / Ddots }
5282 {
5283     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5284     draw-first .value_forbidden:n = true ,
5285 }

```

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

```

5286 \cs_new_protected:Npn \@@_Hspace:
5287 {
5288     \bool_gset_true:N \g_@@_empty_cell_bool
5289     \hspace
5290 }

```

In the environments of `nicematrix`, the command `\multicolumn` is redefined. We will patch the environment `{tabular}` to go back to the previous value of `\multicolumn`.

```

5291 \cs_new_eq:NN \@@_old_multicolumn: \multicolumn

```

The command `\@@_Hdotsfor` will be linked to `\Hdotsfor` in `{NiceArrayWithDelims}`. TikZ nodes are created also in the implicit cells of the `\Hdotsfor` (maybe we should modify that point).

This command must *not* be protected since it begins with `\multicolumn`.

```

5292 \cs_new:Npn \@@_Hdotsfor:
5293 {
5294     \bool_lazy_and:nnTF
5295     { \int_if_zero_p:n \c@jCol }
5296     { \int_if_zero_p:n \l_@@_first_col_int }
5297     {
5298         \bool_if:NTF \g_@@_after_col_zero_bool
5299         {
5300             \multicolumn { 1 } { c } { }
5301             \@@_Hdotsfor_i:
5302         }
5303         { \@@_fatal:n { Hdotsfor~in~col~0 } }
5304     }
5305     {
5306         \multicolumn { 1 } { c } { }
5307         \@@_Hdotsfor_i:
5308     }
5309 }

```

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:`).

```
5310 \AtBeginDocument
5311 {
```

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

```
5312 \cs_new_protected:Npn \@@_Hdotsfor_i:
5313 { \@@_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`).

```
5314 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m O { } E { _ ^ : } { { } { } { } } }
5315 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5316 {
5317   \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5318   {
5319     \@@_Hdotsfor:nnnn
5320     { \int_use:N \c@iRow }
5321     { \int_use:N \c@jCol }
5322     { #2 }
5323     {
5324       #1 , #3 ,
5325       down = \exp_not:n { #4 } ,
5326       up = \exp_not:n { #5 } ,
5327       middle = \exp_not:n { #6 }
5328     }
5329   }
5330   \prg_replicate:nn { #2 - 1 }
5331   {
5332     &
5333     \multicolumn { 1 } { c } { }
5334     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5335   }
5336 }
5337 }
```

```
5338 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5339 {
5340   \bool_set_false:N \l_@@_initial_open_bool
5341   \bool_set_false:N \l_@@_final_open_bool
```

For the row, it's easy.

```
5342 \int_set:Nn \l_@@_initial_i_int { #1 }
5343 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
```

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

```
5344 \int_compare:nNnTF { #2 } = 1
5345 {
5346   \int_set:Nn \l_@@_initial_j_int 1
5347   \bool_set_true:N \l_@@_initial_open_bool
5348 }
5349 {
5350   \cs_if_exist:cTF
5351   {
5352     pgf @ sh @ ns @ \@@_env:
5353     - \int_use:N \l_@@_initial_i_int
5354     - \int_eval:n { #2 - 1 }
5355   }
5356   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5357   {
5358     \int_set:Nn \l_@@_initial_j_int { #2 }
5359     \bool_set_true:N \l_@@_initial_open_bool
```

```

5360     }
5361   }
5362   \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5363   {
5364     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5365     \bool_set_true:N \l_@@_final_open_bool
5366   }
5367   {
5368     \cs_if_exist:cTF
5369     {
5370       pgf @ sh @ ns @ \@@_env:
5371       - \int_use:N \l_@@_final_i_int
5372       - \int_eval:n { #2 + #3 }
5373     }
5374     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5375     {
5376       \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5377       \bool_set_true:N \l_@@_final_open_bool
5378     }
5379   }
5380   \bool_if:NT \g_@@_aux_found_bool
5381   {
5382     {
5383       \@@_open_shorten:
5384       \int_if_zero:nTF { #1 }
5385       { \color { nicematrix-first-row } }
5386       {
5387         \int_compare:nNnT { #1 } = \g_@@_row_total_int
5388         { \color { nicematrix-last-row } }
5389       }
5390       \keys_set:nn { nicematrix / xdots } { #4 }
5391       \@@_color:o \l_@@_xdots_color_tl
5392       \@@_actually_draw_Ldots:
5393     }
5394   }

```

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).

```

5395   \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5396   { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5397 }

```

```

5398 \AtBeginDocument
5399 {
5400   \cs_new_protected:Npn \@@_Vdotsfor:
5401   { \@@_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`).

```

5402   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5403   \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5404   {
5405     \bool_gset_true:N \g_@@_empty_cell_bool
5406     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5407     {
5408       \@@_Vdotsfor:nnnn
5409       { \int_use:N \c@iRow }
5410       { \int_use:N \c@jCol }
5411       { #2 }
5412       {
5413         #1 , #3 ,

```

```

5414         down = \exp_not:n { #4 } ,
5415         up = \exp_not:n { #5 } ,
5416         middle = \exp_not:n { #6 }
5417     }
5418 }
5419 }
5420 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5421 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5422 {
5423     \bool_set_false:N \l_@@_initial_open_bool
5424     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5425     \int_set:Nn \l_@@_initial_j_int { #2 }
5426     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5427     \int_compare:nNnTF { #1 } = 1
5428     {
5429         \int_set:Nn \l_@@_initial_i_int 1
5430         \bool_set_true:N \l_@@_initial_open_bool
5431     }
5432     {
5433         \cs_if_exist:cTF
5434         {
5435             pgf @ sh @ ns @ \@@_env:
5436             - \int_eval:n { #1 - 1 }
5437             - \int_use:N \l_@@_initial_j_int
5438         }
5439         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5440         {
5441             \int_set:Nn \l_@@_initial_i_int { #1 }
5442             \bool_set_true:N \l_@@_initial_open_bool
5443         }
5444     }
5445     \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5446     {
5447         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5448         \bool_set_true:N \l_@@_final_open_bool
5449     }
5450     {
5451         \cs_if_exist:cTF
5452         {
5453             pgf @ sh @ ns @ \@@_env:
5454             - \int_eval:n { #1 + #3 }
5455             - \int_use:N \l_@@_final_j_int
5456         }
5457         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5458         {
5459             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5460             \bool_set_true:N \l_@@_final_open_bool
5461         }
5462     }
5463     \bool_if:NT \g_@@_aux_found_bool
5464     {
5465         {
5466             \@@_open_shorten:
5467             \int_if_zero:nTF { #2 }
5468             { \color { nicematrix-first-col } }

```

```

5469         {
5470             \int_compare:nNtT { #2 } = \g_@@_col_total_int
5471             { \color { nicematrix-last-col } }
5472         }
5473         \keys_set:nn { nicematrix / xdots } { #4 }
5474         \@@_color:o \l_@@_xdots_color_tl
5475         \bool_if:NTF \l_@@_Vbrace_bool
5476             \@@_actually_draw_Vbrace:
5477             \@@_actually_draw_Vdots:
5478     }
5479 }

```

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).

```

5480     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5481     { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5482 }

```

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

```

5483 \NewDocumentCommand \@@_rotate: { 0 { } }
5484 {
5485     \bool_gset_true:N \g_@@_rotate_bool
5486     \keys_set:nn { nicematrix / rotate } { #1 }
5487     \ignorespaces
5488 }

```

The command `\@@_rotate_p_col:` will be linked to `\rotate` in the the cells of the columns of type *p* and *al*.

```

5489 \cs_new_protected:Npn \@@_rotate_p_col: { \@@_error:n { rotate~in~p~col } }

5490 \keys_define:nn { nicematrix / rotate }
5491 {
5492     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5493     c .value_forbidden:n = true ,
5494     -90 .code:n = \bool_gset_true:N \g_@@_rotate_minus_bool ,
5495     -90 .value_forbidden:n = true ,
5496     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5497 }

```

19 The command `\line` accessible in `\CodeAfter`

In the `\CodeAfter`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format *i-j*) and draws a dotted line between these cells. In fact, it also works with names of blocks.

First, we write a command with the following behaviour:

- If the argument is of the format *i-j*, our command applies the command `\int_eval:n` to *i* and *j* ;
- If not (that is to say, when it’s a name of a `\Block`), the argument is left unchanged.

This must *not* be protected (and is, of course fully expandable).¹⁴

```

5498 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5499 {
5500   \tl_if_empty:nTF { #2 }
5501     { #1 }
5502     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5503 }
5504 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5505 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

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

```

5506 \AtBeginDocument
5507 {

```

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

```

5508   \tl_set_rescan:Nnn \l_tmpa_tl { }
5509   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5510   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5511     {
5512       {
5513         \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5514         \@@_color:o \l_@@_xdots_color_tl
5515         \use:e
5516         {
5517           \@@_line_i:nn
5518             { \@@_double_int_eval:n #2 - \q_stop }
5519             { \@@_double_int_eval:n #3 - \q_stop }
5520         }
5521       }
5522     }
5523 }

5524 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5525 {
5526   \bool_set_false:N \l_@@_initial_open_bool
5527   \bool_set_false:N \l_@@_final_open_bool
5528   \bool_lazy_or:nnTF
5529     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5530     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5531     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

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

```

5532   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5533 }

5534 \AtBeginDocument
5535 {
5536   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5537   {

```

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:`.

```

5538   \c_@@_pgfortikzpicture_tl
5539   \@@_draw_line_iii:nn { #1 } { #2 }
5540   \c_@@_endpgfortikzpicture_tl
5541 }
5542 }

```

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

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

```

5543 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5544 {
5545   \pgfrememberpicturepositiononpagetrue
5546   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5547   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5548   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5549   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5550   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5551   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5552   \@@_draw_line:
5553 }

```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don't use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

20 The command `\RowStyle`

`\g_@@_row_style_tl` may contain several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

Then, `\g_@@_row_style_tl` will be inserted in all the cells of the array (and also in both components of a `\diagbox` in a cell of in a mono-row block).

The test `\@@_if_row_less_then:nn` ensures that the instructions are inserted only if you are in a row which is (still) in the scope of that instructions (which depends on the value of the key `nb-rows` of `\RowStyle`).

That test will be active even in an expandable context because `\@@_if_row_less_then:nn` is *not* protected.

`#1` is the first row *after* the scope of the instructions in `#2`

However, both arguments are implicit because they are taken by curryfication.

```

5554 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNnT \c@iRow < }
5555 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNnF \c@jCol < }

```

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

```

5556 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5557 {
5558   \tl_gput_right:Ne \g_@@_row_style_tl
5559   {

```

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

```

5560   \exp_not:N
5561   \@@_if_row_less_than:nn
5562   { \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`).

```

5563   {
5564     \exp_not:N
5565     \@@_if_col_greater_than:nn
5566     { \int_use:N \c@jCol }
5567     { \exp_not:n { #1 } \scan_stop: }
5568   }
5569 }
5570 }
5571 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

```

```

5572 \keys_define:nn { nicematrix / RowStyle }
5573 {
5574   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5575   cell-space-top-limit+ .code:n =
5576     \dim_set:Nn \l_tmpa_dim { \l_@@_cell_space_top_limit_dim + #1 } ,
5577   cell-space-top-limit+ .value_required:n = true ,
5578   cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
5579   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5580   cell-space-bottom-limit+ .code:n =
5581     \dim_set:Nn \l_tmpb_dim { \l_@@_cell_space_bottom_limit_dim + #1 } ,
5582   cell-space-bottom-limit+ .value_required:n = true ,
5583   cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
5584   cell-space-limits .meta:n =
5585     {
5586       cell-space-top-limit = #1 ,
5587       cell-space-bottom-limit = #1 ,
5588     } ,
5589   cell-space-limits+ .meta:n =
5590     {
5591       cell-space-top-limit += #1 ,
5592       cell-space-bottom-limit += #1 ,
5593     } ,
5594   cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
5595   color .tl_set:N = \l_@@_color_tl ,
5596   color .value_required:n = true ,
5597   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5598   nb-rows .code:n =
5599     \str_if_eq:eeTF { #1 } { * }
5600     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5601     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5602   nb-rows .value_required:n = true ,
5603   fill .tl_set:N = \l_@@_fill_tl ,
5604   fill .value_required:n = true ,

```

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

```

5605   opacity .tl_set:N = \l_@@_opacity_tl ,
5606   opacity .value_required:n = true ,
5607   rowcolor .tl_set:N = \l_@@_fill_tl ,
5608   rowcolor .value_required:n = true ,
5609   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5610   rounded-corners .default:n = 4 pt ,
5611   unknown .code:n =
5612     \@@_unknown_key:nn
5613     { nicematrix / RowStyle }
5614     { Unknown-key-for-RowStyle }
5615 }

```

```

5616 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5617 {
5618   \group_begin:
5619   \tl_clear:N \l_@@_fill_tl
5620   \tl_clear:N \l_@@_opacity_tl
5621   \tl_clear:N \l_@@_color_tl
5622   \int_set:Nn \l_@@_key_nb_rows_int 1
5623   \dim_zero:N \l_@@_rounded_corners_dim
5624   \dim_zero:N \l_tmpa_dim
5625   \dim_zero:N \l_tmpb_dim
5626   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5627   \tl_if_empty:NF \l_@@_fill_tl
5628   {
5629     \@@_add_opacity_to_fill:

```

```

5630 \tl_gput_right:Ne \g_@@_pre_code_before_tl
5631 {
The command \@@_exp_color_arg:No is fully expandable.
5632 \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5633 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5634 {
5635 \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5636 - *
5637 }
5638 { \dim_use:N \l_@@_rounded_corners_dim }
5639 }
5640 }
5641 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5642 \dim_compare:nNt \l_tmpa_dim > \c_zero_dim
5643 {
5644 \@@_put_in_row_style:e
5645 {
5646 \@@_put_in_cell_after_hook:n
5647 {

```

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

```

5648 \dim_set:Nn \l_@@_cell_space_top_limit_dim
5649 { \dim_use:N \l_tmpa_dim }
5650 }
5651 }
5652 }

```

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

```

5653 \dim_compare:nNt \l_tmpb_dim > \c_zero_dim
5654 {
5655 \@@_put_in_row_style:e
5656 {
5657 \@@_put_in_cell_after_hook:n
5658 {
5659 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5660 { \dim_use:N \l_tmpb_dim }
5661 }
5662 }
5663 }

```

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

```

5664 \tl_if_empty:NF \l_@@_color_tl
5665 {
5666 \@@_put_in_row_style:e
5667 {
5668 \mode_leave_vertical:
5669 \@@_color:n { \l_@@_color_tl }
5670 }
5671 }

```

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

```

5672 \bool_if:NT \l_@@_bold_row_style_bool
5673 {
5674 \@@_put_in_row_style:n
5675 {
5676 \exp_not:n
5677 {
5678 \if_mode_math:
5679 $ % $
5680 \bfseries \boldmath
5681 $ % $
5682 \else:
5683 \bfseries \boldmath

```

```

5684         \fi:
5685     }
5686 }
5687 }
5688 \group_end:
5689 \g_@@_row_style_tl
5690 \ignorespaces
5691 }

```

The following commande must *not* be protected.

```

5692 \cs_new:Npn \@@_rounded_from_row:n #1
5693 {
5694     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

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

```

5695     { \int_eval:n { #1 } - 1 }
5696     {
5697         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5698         - \exp_not:n { \int_use:N \c@jCol }
5699     }
5700     { \dim_use:N \l_@@_rounded_corners_dim }
5701 }

```

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).

```

5702 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5703 {

```

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.

```

5704     \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`.

```

5705     \str_if_in:nnF { #1 } { !! }
5706     {
5707         \seq_map_indexed_inline:Nn \g_@@_colors_seq

```

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

```

5708     { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5709   }
5710   \int_if_zero:nTF \l_tmpa_int

```

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

```

5711   {
5712     \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5713     \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5714   }

```

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

```

5715     { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5716   }
5717   \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }

```

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

```

5718   \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5719   {
5720     \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5721     {

```

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

```

5722     \group_begin:
5723     \pgfsetcornersarced
5724     { \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.

```

5725     \bool_if:NTF \l_@@_hvlines_bool
5726     {
5727       \pgfpathrectanglecorners
5728       {
5729         \pgfpointadd
5730         { \@@_qpoint:n { row-1 } }
5731         { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
5732       }
5733       {
5734         \pgfpointadd
5735         {
5736           \@@_qpoint:n
5737           { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
5738         }
5739         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5740       }
5741     }
5742     {
5743       \pgfpathrectanglecorners
5744       { \@@_qpoint:n { row-1 } }
5745       {
5746         \pgfpointadd
5747         {
5748           \@@_qpoint:n
5749           { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
5750         }
5751         { \pgfpoint \c_zero_dim \arrayrulewidth }
5752       }
5753     }
5754     \pgfusepath { clip }
5755     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```
5756     }
5757 }
```

The command `\@@_actually_color:` will actually fill the rectangles, color by color, as specified in the sequence `\g_@@_colors_seq`.

```
5758 \cs_new_protected:Npn \@@_actually_color:
5759 {
5760   \pgfpicture
5761   \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.

```
5762   \@@_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`).

```
5763   \seq_map_indexed_inline:Nn \g_@@_colors_seq
5764   {
5765     \int_compare:nNnTF { ##1 } = 1
5766     {
5767       \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5768       \use:c { g_@@_color _ 1 _tl }
5769       \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5770     }
5771     {
5772       \pgfscope
5773       \@@_color_opacity: ##2
5774       \use:c { g_@@_color _ ##1 _tl }
5775       \tl_gclear:c { g_@@_color _ ##1 _tl }
5776       \pgfusepath { fill }
5777       \endpgfscope
5778     }
5779   }
5780 \endpgfpicture
5781 }
```

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

```
5782 \cs_new_protected:Npn \@@_color_opacity:
5783 {
5784   \peek_meaning:NTF [
5785     \@@_color_opacity:w
5786     { \@@_color_opacity:w [ ] }
5787   }
```

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.

```
5788 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5789 {
5790   \tl_clear:N \l_tmpa_tl
5791   \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.

```
5792   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5793   \tl_if_empty:NTF \l_tmpb_tl
5794     \@declaredcolor
5795     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5796 }
```

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

```

5797 \keys_define:nn { nicematrix / color-opacity }
5798 {
5799   opacity .tl_set:N          = \l_tmpa_tl ,
5800   opacity .value_required:n = true
5801 }

```

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

```

5802 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5803 {
5804   \def \l_@@_rows_tl { #1 }
5805   \def \l_@@_cols_tl { #2 }
5806   \@@_cartesian_path:
5807 }

```

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

```

5808 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5809 {
5810   \tl_if_blank:nF { #2 }
5811   {
5812     \@@_add_to_colors_seq:en
5813     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5814     { \@@_cartesian_color:nn { #3 } { - } }
5815   }
5816 }

```

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

```

5817 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5818 {
5819   \tl_if_blank:nF { #2 }
5820   {
5821     \@@_add_to_colors_seq:en
5822     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5823     { \@@_cartesian_color:nn { - } { #3 } }
5824   }
5825 }

```

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

```

5826 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5827 {
5828   \tl_if_blank:nF { #2 }
5829   {
5830     \@@_add_to_colors_seq:en
5831     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5832     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5833   }
5834 }

```

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

```

5835 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5836 {
5837   \tl_if_blank:nF { #2 }
5838   {
5839     \@@_add_to_colors_seq:en
5840     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5841     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5842   }
5843 }

```

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

```

5844 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5845 {
5846   \@@_cut_on_hyphen:w #1 \q_stop
5847   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5848   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5849   \@@_cut_on_hyphen:w #2 \q_stop
5850   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5851   \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`.

```

5852   \@@_cartesian_path:n { #3 }
5853 }

```

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

```

5854 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5855 {
5856   \clist_map_inline:nn { #3 }
5857     { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5858 }

```

```

5859 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5860 {
5861   \int_step_inline:nn \c@iRow
5862   {
5863     \int_step_inline:nn \c@jCol
5864     {
5865       \int_if_even:nTF { ####1 + ##1 }
5866         { \@@_cellcolor [ #1 ] { #2 } }
5867         { \@@_cellcolor [ #1 ] { #3 } }
5868       { ##1 - ####1 }
5869     }
5870   }
5871 }

```

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”.

```

5872 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5873 {
5874   \@@_rectanglecolor [ #1 ] { #2 }
5875   { 1 - 1 }
5876   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5877 }

5878 \keys_define:nn { nicematrix / rowcolors }
5879 {
5880   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5881   cols .tl_set:N = \l_@@_cols_tl ,
5882   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5883   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5884 }

```

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*.

```
5885 \NewDocumentCommand \l_@@_rowlistcolors { 0 { } m m 0 { } }
5886 {
```

\l_@@_colors_seq will be the list of colors.

```
5887 \seq_clear_new:N \l_@@_colors_seq
5888 \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5889 \tl_clear_new:N \l_@@_cols_tl
5890 \tl_set:Nn \l_@@_cols_tl { - }
5891 \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).

```
5892 \int_zero_new:N \l_@@_color_int
5893 \int_set:Nn \l_@@_color_int 1
5894 \bool_if:NT \l_@@_respect_blocks_bool
5895 {
```

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).

```
5896 % modified 2026-02-18
5897 \seq_set_filter:NNn \l_tmpa_seq \g_@@_pos_of_blocks_seq
5898 { \@@_not_in_exterior_p:nnnnn ##1 }
5899 }
```

#2 is the list of intervals of rows.

```
5900 \clist_map_inline:nn { #2 }
5901 {
5902 \tl_set:Nn \l_tmpa_tl { ##1 }
5903 \tl_if_in:NnTF \l_tmpa_tl { - }
5904 { \@@_cut_on_hyphen:w ##1 \q_stop }
5905 { \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.

```
5906 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5907 \int_set:Nn \l_@@_color_int
5908 { \bool_if:NNTF \l_@@_rowcolors_restart_bool { 1 } \l_tmpa_tl }
5909 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5910 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5911 {
```

We will compute in \l_tmpb_int the last row of the “block”.

```
5912 \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).

```
5913 \bool_if:NT \l_@@_respect_blocks_bool
5914 {
5915 \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5916 { \@@_intersect_our_row_p:nnnnn #####1 }
5917 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }
```

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

```
5918 }
5919 \tl_set:Nn \l_@@_rows_tl
5920 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
```

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

```
5921 \tl_set:Nn \l_@@_color_tl
5922 {
5923 \@@_color_index:n
5924 {
5925 \int_mod:nn
5926 { \l_@@_color_int - 1 }
5927 { \seq_count:N \l_@@_colors_seq }
```

```

5928         + 1
5929     }
5930 }
5931 \tl_if_empty:NF \l_@@_color_tl
5932 {
5933     \use:e
5934     {
5935         \@@_add_to_colors_seq:nn
5936         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5937         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5938     }
5939 }
5940 \int_incr:N \l_@@_color_int
5941 \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5942 }
5943 }
5944 }

```

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.

```

5945 \cs_new:Npn \@@_color_index:n #1
5946 {

```

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

```

5947     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5948     { \@@_color_index:n { #1 - 1 } }
5949     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5950 }

```

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.

```

5951 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5952 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5953 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5954 {
5955     \int_compare:nNnT { #3 } > \l_tmpb_int
5956     { \int_set:Nn \l_tmpb_int { #3 } }
5957 }

```

```

5958 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5959 {
5960     \int_if_zero:nTF { #4 }
5961     \prg_return_false:
5962     {
5963         \int_compare:nNnTF { #2 } > \c@jCol
5964         \prg_return_false:
5965         \prg_return_true:
5966     }
5967 }

```

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

```

5968 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5969 {
5970     \int_compare:nNnTF { #1 } > \l_tmpa_int
5971     \prg_return_false:
5972     {
5973         \int_compare:nNnTF \l_tmpa_int > { #3 }

```

```

5974         \prg_return_false:
5975         \prg_return_true:
5976     }
5977 }

```

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`).

```

5978 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5979 {
5980     \dim_compare:nNnTF { #1 } = \c_zero_dim
5981     {
5982         \bool_if:NTF \l_@@_nocolor_used_bool
5983         { \@@_cartesian_path_normal_ii: }
5984         {
5985             \clist_if_empty:NTF \l_@@_corners_cells_clist
5986             { \@@_cartesian_path_normal_i:n { #1 } }
5987             { \@@_cartesian_path_normal_ii: }
5988         }
5989     }
5990     { \@@_cartesian_path_normal_i:n { #1 } }
5991 }

```

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.

```

5992 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5993 {
5994     \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5995     \clist_map_inline:Nn \l_@@_cols_tl
5996     {

```

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

```

5997         \def \l_tmpa_tl { ##1 }
5998         \tl_if_in:NnTF \l_tmpa_tl { - }
5999         { \@@_cut_on_hyphen:w ##1 \q_stop }
6000         { \def \l_tmpb_tl { ##1 } }
6001         \tl_if_empty:NTF \l_tmpa_tl
6002         { \def \l_tmpa_tl { 1 } }
6003         {
6004             \str_if_eq:eeT \l_tmpa_tl { * }
6005             { \def \l_tmpa_tl { 1 } }
6006         }
6007         \int_compare:nNnT \l_tmpa_tl > \g_@@_col_total_int
6008         { \@@_error:n { Invalid~col~number } }
6009         \tl_if_empty:NTF \l_tmpb_tl
6010         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6011         {
6012             \str_if_eq:eeT \l_tmpb_tl { * }
6013             { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6014         }
6015         \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
6016         { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

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

```

6017         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6018         \@@_qpoint:n { col - \l_tmpa_tl }
6019         \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
6020         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }

```

```

6021     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6022     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } } }
6023     \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.

```

6024     \clist_map_inline:Nn \l_@@_rows_tl
6025     {
6026         \def \l_tmpa_tl { #####1 }
6027         \tl_if_in:NnTF \l_tmpa_tl { - }
6028         { \@@_cut_on_hyphen:w #####1 \q_stop }
6029         { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
6030         \tl_if_empty:NnTF \l_tmpa_tl
6031         { \def \l_tmpa_tl { 1 } }
6032         {
6033             \str_if_eq:eeT \l_tmpa_tl { * }
6034             { \def \l_tmpa_tl { 1 } }
6035         }
6036         \tl_if_empty:NnTF \l_tmpb_tl
6037         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6038         {
6039             \str_if_eq:eeT \l_tmpb_tl { * }
6040             { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6041         }
6042         \int_compare:nNnT \l_tmpa_tl > \g_@@_row_total_int
6043         { \@@_error:n { Invalid~row~number } } }
6044         \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
6045         { \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`.

```

6046         \cs_if_exist:cF
6047         { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
6048         {
6049             \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
6050             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6051             \@@_qpoint:n { row - \l_tmpa_tl }
6052             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6053             \pgfpathrectanglecorners
6054             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6055             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6056         }
6057     }
6058 }
6059 }

```

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

```

6060 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
6061 {
6062     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6063     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6064     \clist_map_inline:Nn \l_@@_cols_tl
6065     {
6066         \@@_qpoint:n { col - ##1 }
6067         \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
6068         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6069         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6070         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } } }
6071     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

6072     \clist_map_inline:Nn \l_@@_rows_tl
6073     {
6074         \@@_if_in_corner:nF { #####1 - ##1 }

```

```

6075     {
6076         \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
6077         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6078         \@@_qpoint:n { row - #####1 }
6079         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6080         \cs_if_exist:cF { @@ _ nocolor _ #####1 - #1 }
6081         {
6082             \pgfpathrectanglecorners
6083             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6084             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6085         }
6086     }
6087 }
6088 }
6089 }

```

The following command corresponds to a radius of the corners equal to 0 pt. This command is used by the commands `\@@_rowcolors`, `\@@_columncolor` and `\@@_rowcolor:n` (used in `\@@_rowcolor`).

```

6090 \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

```

6091 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
6092 {
6093     \bool_set_true:N \l_@@_nocolor_used_bool
6094     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6095     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6096     \clist_map_inline:Nn \l_@@_rows_tl
6097     {
6098         \clist_map_inline:Nn \l_@@_cols_tl
6099         { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
6100     }
6101 }

```

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.

```

6102 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
6103 {
6104     \clist_set_eq:NN \l_tmpa_clist #1
6105     \clist_clear:N #1
6106     \clist_map_inline:Nn \l_tmpa_clist
6107     {

```

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

```

6108     \def \l_tmpa_tl { ##1 }
6109     \tl_if_in:NnTF \l_tmpa_tl { - }
6110     { \@@_cut_on_hyphen:w ##1 \q_stop }
6111     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
6112     \bool_lazy_or:nnT
6113     { \str_if_eq_p:ee \l_tmpa_tl { * } }
6114     { \tl_if_blank_p:o \l_tmpa_tl }
6115     { \def \l_tmpa_tl { 1 } }
6116     \bool_lazy_or:nnT
6117     { \str_if_eq_p:ee \l_tmpb_tl { * } }
6118     { \tl_if_blank_p:o \l_tmpb_tl }
6119     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6120     \int_compare:nNnT \l_tmpb_tl > { #2 }
6121     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }

```

```

6122     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
6123     { \clist_put_right:Nn #1 { #####1 } }
6124   }
6125 }

```

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

```

6126 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
6127 {
6128   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6129   {

```

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`).

```

6130     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
6131     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6132   }
6133   \ignorespaces
6134 }

6135 \NewDocumentCommand \@@_cellcolor_error { 0 { } m }
6136 { \@@_error:n { cellcolor~in~Block } }
6137 % \end{macrocode}
6138 %
6139 % \begin{macrocode}
6140 \NewDocumentCommand \@@_rowcolor_error { 0 { } m }
6141 { \@@_error:n { rowcolor~in~Block } }
6142 % \end{macrocode}
6143 %
6144 % \bigskip
6145 % The following command will be linked to |\rowcolor| in the tabular.
6146 % \begin{macrocode}
6147 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
6148 {
6149   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6150   {
6151     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
6152     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6153     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
6154   }
6155   \ignorespaces
6156 }

```

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

```

6157 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
6158 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

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

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

```

6159 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
6160 {

```

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`.

```

6161   \seq_gclear:N \g_tmpa_seq
6162   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6163   { \@@_rowlistcolors_tabular:nnnn #1 }
6164   \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`.

```

6165 \seq_gput_right:Ne \g_@@_rowlistcolors_seq
6166 {
6167   { \int_use:N \c@iRow }
6168   { \exp_not:n { #1 } }
6169   { \exp_not:n { #2 } }
6170   { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
6171 }
6172 \ignorespaces
6173 }

```

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`).

```

6174 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
6175 {
6176   \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).

```

6177   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
6178   {
6179     \tl_gput_right:Ne \g_@@_pre_code_before_tl
6180     {
6181       \@@_rowlistcolors
6182       [ \exp_not:n { #2 } ]
6183       { #1 - \int_eval:n { \c@iRow - 1 } }
6184       { \exp_not:n { #3 } }
6185       [ \exp_not:n { #4 } ]
6186     }
6187   }
6188 }

```

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.

```

6189 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
6190 {
6191   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6192   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
6193   \seq_gclear:N \g_@@_rowlistcolors_seq
6194 }

```

```

6195 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6196 {
6197   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6198   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6199 }

```

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.

```

6200 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
6201 {

```

With the following line, we test whether the cell is the first one we encounter in its column (don't forget that some rows may be incomplete).

```
6202 \int_compare:nNtT \c@jCol > \g_@@_col_total_int
6203 {
```

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).

```
6204 \tl_gput_left:Nx \g_@@_pre_code_before_tl
6205 {
6206   \exp_not:N \columncolor [ #1 ]
6207   { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6208 }
6209 }
6210 }

6211 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6212 {
6213   \clist_map_inline:nn { #1 }
6214   {
6215     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6216     { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6217     \columncolor { nocolor } { ##1 }
6218   }
6219 }

6220 \cs_new_protected:Npn \@@_EmptyRow:n #1
6221 {
6222   \clist_map_inline:nn { #1 }
6223   {
6224     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6225     { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6226     \rowcolor { nocolor } { ##1 }
6227   }
6228 }
```

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`.

```
6229 \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`.

```
6230 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6231 {
6232   \int_if_zero:nTF \l_@@_first_col_int
6233   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6234   {
```



```

6235 \int_if_zero:nTF \c@jCol
6236 {
6237     \int_compare:nNf \c@iRow = { -1 }
6238     {
6239         \int_compare:nNf \c@iRow = { \l_@@_last_row_int - 1 }
6240         { #1 }
6241     }
6242 }
6243 { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6244 }
6245 }

```

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.

```

6246 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6247 {
6248     \int_if_zero:nF \c@iRow
6249     {
6250         \int_compare:nNf \c@iRow = \l_@@_last_row_int
6251         {
6252             \int_compare:nNt \c@jCol > \c_zero_int
6253             { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6254         }
6255     }
6256 }

```

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:nNt \c@iRow < \l_@@_last_row_int`).

The following command will be used for `\Toprule`, `\BottomRule` and `\MidRule`.

```

6257 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6258 {
6259     \IfPackageLoadedTF { tikz }
6260     {
6261         \IfPackageLoadedTF { booktabs }
6262         { #2 }
6263         { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6264     }
6265     { \@@_error:nn { TopRule~without~tikz } { #1 } }
6266 }
6267 \NewExpandableDocumentCommand { \@@_TopRule } { }
6268 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
6269 \cs_new:Npn \@@_TopRule_i:
6270 {
6271     \noalign \bgroup
6272     \peek_meaning:Ntf [
6273     { \@@_TopRule_ii: }
6274     { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6275 }
6276 \NewDocumentCommand \@@_TopRule_ii: { o }
6277 {
6278     \tl_gput_right:Ne \g_@@_rules_tl
6279     {
6280         \@@_draw_hruler:n
6281         {
6282             position = \int_eval:n { \c@iRow + 1 } ,
6283             tikz =
6284             {
6285                 line~width = #1 ,
6286                 yshift = 0.25 \arrayrulewidth ,

```

```

6287         shorten~< = - 0.5 \arrayrulewidth
6288     } ,
6289     total-width = #1
6290 }
6291 }
6292 \skip_vertical:n { \belowrulesep + #1 }
6293 \egroup
6294 }
6295 \NewExpandableDocumentCommand { \@@_BottomRule } { }
6296 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }
6297 \cs_new:Npn \@@_BottomRule_i:
6298 {
6299     \noalign \bgroup
6300     \peek_meaning:NTF [
6301         { \@@_BottomRule_ii: }
6302         { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6303     }
6304 \NewDocumentCommand \@@_BottomRule_ii: { o }
6305 {
6306     \tl_gput_right:Ne \g_@@_rules_tl
6307     {
6308         \@@_draw_hrulerule:n
6309         {
6310             position = \int_eval:n { \c@iRow + 1 } ,
6311             tikz =
6312             {
6313                 line~width = #1 ,
6314                 yshift = 0.25 \arrayrulewidth ,
6315                 shorten~< = - 0.5 \arrayrulewidth
6316             } ,
6317             total-width = #1 ,
6318         }
6319     }
6320     \skip_vertical:N \aboverulesep
6321     \@@_create_row_node_i:
6322     \skip_vertical:n { #1 }
6323     \egroup
6324 }
6325 \NewExpandableDocumentCommand { \@@_MidRule } { }
6326 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6327 \cs_new:Npn \@@_MidRule_i:
6328 {
6329     \noalign \bgroup
6330     \peek_meaning:NTF [
6331         { \@@_MidRule_ii: }
6332         { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6333     }
6334 \NewDocumentCommand \@@_MidRule_ii: { o }
6335 {
6336     \skip_vertical:N \aboverulesep
6337     \@@_create_row_node_i:
6338     \tl_gput_right:Ne \g_@@_rules_tl
6339     {
6340         \@@_draw_hrulerule:n
6341         {
6342             position = \int_eval:n { \c@iRow + 1 } ,
6343             tikz =
6344             {
6345                 line~width = #1 ,
6346                 yshift = 0.25 \arrayrulewidth ,
6347                 shorten~< = - 0.5 \arrayrulewidth

```

```

6348         } ,
6349         total-width = #1 ,
6350     }
6351 }
6352 \skip_vertical:n { \belowrulesep + #1 }
6353 \egroup
6354 }

```

General system for drawing rules

```

6355 \AtBeginDocument
6356 {
6357   \cs_new_protected:Npe \@@_draw_rules:
6358   {
6359     \c_@@_pgfortikzpicture_tl
6360     \@@_draw_rules_i:
6361     \c_@@_endpgfortikzpicture_tl
6362   }
6363 }
6364 \cs_new_protected:Npn \@@_draw_rules_i:
6365 {
6366   \bool_lazy_all:nT
6367   {
6368     { \clist_if_empty_p:N \l_@@_corners_clist }
6369     { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
6370     { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
6371     { \seq_if_empty_p:N \g_@@_pos_of_stroken_blocks_seq }
6372   }
6373   {
6374     \socket_assign_plug:nn { nicematrix / draw-hrule } { quick }
6375     \socket_assign_plug:nn { nicematrix / draw-vrule } { quick }
6376   }
6377   \pgfrememberpicturerepositiononpagetrue
6378   \pgf@relevantforpicturesizefalse
6379   \pgfsetrectcap
6380   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6381   \CT@arc@
6382   \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_key_hlines:
6383   \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_key_vlines:
6384   \g_@@_rules_tl
6385 }

```

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”.

```

6386 \keys_define:nn { nicematrix / rules-after }
6387 {
6388   position .int_set:N = \l_@@_position_int ,
6389   start .int_set:N = \l_@@_start_int ,
6390   start .initial:n = 1 ,
6391   end .int_set:N = \l_@@_end_int ,
6392   multiplicity .code:n = \@@_set_multiplicity:n { #1 } ,
6393   dotted .code:n =
6394     \bool_set_true:N \l_@@_dotted_bool
6395     \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
6396     \socket_assign_plug:nn { nicematrix / hsegment } { dotted } ,
6397   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]`.

```

6398     color .code:n =
6399         \@@_set_CTarc:n { #1 }
6400         \CT@arc@
6401         \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6402     color .value_required:n = true ,
6403     sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6404     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
    }
}

6405     total-width .dim_set:N = \l_@@_rule_width_after_dim ,
6406     unknown .code:n =
6407         \@@_unknown_key:nn
6408             { nicematrix / rules-after }
6409             { Unknown-key-for-a-rule } ,
6410     tikz .value_required:n = true
6411 }

```

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.

```

6412 \AtBeginDocument
6413 {
6414     \IfPackageLoadedTF { tikz }
6415     {
6416         \keys_define:nn { nicematrix / rules-after }
6417         {
6418             tikz .code:n =
6419                 \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 }
6420                 \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
6421                 \socket_assign_plug:nn { nicematrix / hsegment } { tikz }
6422         }
6423     }
6424     {
6425         \keys_define:nn { nicematrix / rules-after }
6426         { tikz .code:n = \@@_error:n { tikz~without~tikz } }
6427     }
6428 }

6429 \cs_new_protected:Npn \@@_set_multiplicity:n #1
6430 {
6431     \int_set:Nn \l_@@_multiplicity_int { #1 }
6432     \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
6433     \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
6434 }

```

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.

```
6435 \cs_new_protected:Npn \@@_draw_vrule:n #1
6436 {
```

The group is for the options.

```
6437 {
6438   \int_set_eq:NN \l_@@_end_int \c@iRow
6439   \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).

```
6440   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6441   \socket_use:n { nicematrix / draw-vrule }
6442 }
6443 }
```

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.

```
6444 \socket_new:nn { nicematrix / draw-vrule } { 0 }
6445 \socket_new_plug:nnn { nicematrix / draw-vrule } { general }
6446 {
6447   \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`.

```
6448   \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).

```
6449   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6450   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6451   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6452   \l_tmpa_tl
6453   {
```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```
6454   \@@_test_v:
6455   \bool_if:NTF \g_tmpa_bool
6456   {
6457     \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.

```
6458     { \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl }
6459     { \socket_use:nn { nicematrix / draw-at-odd-vertex-v } }
6460   }
6461   {
6462     \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6463     {
6464       \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }
```

The socket `vsegment` is defined below with its four plugs.

```

6465         \socket_use:n { nicematrix / vsegment }
6466         \int_zero:N \l_@@_segment_start_int
6467     }
6468 }
6469 }
6470 \int_compare:nNt \l_@@_segment_start_int > \c_zero_int
6471 {
6472     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6473     \socket_use:n { nicematrix / vsegment }
6474 }
6475 \group_end:
6476 }

6477 \socket_assign_plug:nn { nicematrix / draw-vrule } { general }
6478 \socket_new_plug:nnn { nicematrix / draw-vrule } { quick }
6479 {
6480     \group_begin:
6481     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6482     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6483     \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
6484     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6485     \socket_use:n { nicematrix / vsegment }
6486     \group_end:
6487 }

```

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.

```

6488 \cs_new_protected:Npn \@@_test_v:
6489 {
6490     \bool_gset_true:N \g_tmpa_bool
6491     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6492     \bool_if:NT \g_tmpa_bool
6493     {
6494         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6495         { \@@_test_vline_in_block:nnnnn ##1 }
6496         \bool_if:NT \g_tmpa_bool
6497         {
6498             \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6499             { \@@_test_vline_in_block:nnnnn ##1 }
6500             \bool_if:NT \g_tmpa_bool
6501             {
6502                 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6503                 { \@@_test_vline_in_stroken_block:nnnn ##1 }
6504             }
6505         }
6506     }
6507 }

6508 \socket_new:nn { nicematrix / draw-at-odd-vertex-v } { 0 }
6509 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-v } { active }
6510 {
6511     {
6512         \int_compare:nNtF \l_@@_position_int > \c@jCol
6513         { \bool_set_false:N \l_tmpa_bool }
6514         {
6515             \@@_test_h:
6516             \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6517         }
6518         \int_compare:nNtF \l_@@_position_int = 1
6519         { \bool_gset_false:N \g_tmpa_bool }
6520         {

```

```

6521         \tl_set:Nn \l_tmpb_tl { \int_eval:n { \l_tmpb_tl - 1 } }
6522         \@@_test_h:
6523     }
6524     \bool_gset:Nn \g_tmpa_bool
6525     { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6526 }
6527 \bool_if:NT \g_tmpa_bool
6528 {
6529     \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }
6530     \socket_use:n { nicematrix / vsegment }
6531     \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl
6532 }
6533 }

6534 \cs_new_protected:Npn \@@_test_in_corner_v:
6535 {
6536     \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6537     {
6538         \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6539         { \bool_set_false:N \g_tmpa_bool }
6540     }
6541     {
6542         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6543         {
6544             \int_compare:nNnTF \l_tmpb_tl = 1
6545             { \bool_set_false:N \g_tmpa_bool }
6546             {
6547                 \@@_if_in_corner:nT
6548                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6549                 { \bool_set_false:N \g_tmpa_bool }
6550             }
6551         }
6552     }
6553 }

```

For the drawing of the (vertical) segments, we will use a socket with four plugs :

- a plug `standar` for the simple rules.
- a plug `multiple` for the rules similar to the standard rules of `array` and `colortbl`, that is to say with multiplicity, etc;
- a plug `dotted` for the rules with our own system of dotted rules;
- a plug `tikz` for the rules drawn with TikZ.

```

6554 \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`.

```

6555 \socket_new_plug:nnn { nicematrix / vsegment } { standard }
6556 {
6557     \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6558     \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6559     \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6560     \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6561     \pgfusepathqstroke
6562 }
6563 \socket_assign_plug:nn { nicematrix / vsegment } { standard }

```

```

6564 \socket_new_plug:nnn { nicematrix / vsegment } { multiple }
6565 {
6566   \dim_add:Nn \l_@@_x_initial_dim
6567   {
6568     - 0.5 \l_@@_rule_width_after_dim
6569     +
6570     ( \arrayrulewidth * \l_@@_multiplicity_int
6571       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6572   }
6573   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6574   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6575   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6576   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6577   \bool_lazy_and:nnT
6578   { \cs_if_exist_p:N \CT@drsc@ }
6579   { ! \tl_if_blank_p:o \CT@drsc@ }
6580   {
6581     {
6582       \CT@drsc@
6583       \dim_add:Nn \l_@@_y_initial_dim { 0.5 \arrayrulewidth }
6584       \dim_sub:Nn \l_@@_y_final_dim { 0.5 \arrayrulewidth }
6585       \dim_set:Nn \l_@@_tmpd_dim
6586       {
6587         \l_@@_x_initial_dim - ( \doublerulesep + \arrayrulewidth )
6588         * ( \l_@@_multiplicity_int - 1 )
6589       }
6590       \pgfpathrectanglecorners
6591       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6592       { \pgfpoint \l_@@_tmpd_dim \l_@@_y_final_dim }
6593       \pgfusepath { fill }
6594     }
6595   }
6596   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6597   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6598   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6599   {
6600     \dim_sub:Nn \l_@@_x_initial_dim { \arrayrulewidth + \doublerulesep }
6601     \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6602     \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6603   }
6604   \pgfusepathqstroke
6605 }

6606 \socket_new_plug:nnn { nicematrix / vsegment } { dotted }
6607 {
6608   \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6609   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6610   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6611   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6612   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6613   \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`


```

6614 \@@_draw_line:
6615 }

6616 \socket_new_plug:nnn { nicematrix / vsegment } { tikz }
6617 {
6618 {

```

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).

```

6619 \tl_if_empty:NF \l_@@_rule_color_tl
6620 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6621 \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6622 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6623 \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6624 \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }

```

The following line can't be short-cutted.

```

6625 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6626 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6627 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim ) --
6628 ( \l_@@_x_initial_dim , \l_@@_y_final_dim ) ;
6629 }
6630 }

```

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).

```

6631 \cs_new_protected:Npn \@@_draw_key_vlines:
6632 {
6633 {
6634 \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
6635 \int_set:Nn \l_@@_end_int \c@iRow

```

There is a currrification in the following code.

```

6636 \str_if_eq:eeTF \l_@@_vlines_clist { all }
6637 { \@@_lines_step_inline:n \c@jCol }
6638 { \clist_map_inline:Nn \l_@@_vlines_clist }
6639 {
6640 \int_set:Nn \l_@@_position_int { ##1 }
6641 \socket_use:n { nicematrix / draw-vrule }
6642 }
6643 }
6644 }

```

The following command is used in `\@@_draw_key_vlines:` and in `\@@_draw_key_hlines:`.

```

6645 \cs_new_protected:Npn \@@_lines_step_inline:n #1
6646 {
6647 \int_step_inline:nnn
6648 { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6649 {
6650 \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6651 { #1 }
6652 { \int_eval:n { #1 + 1 } }
6653 }
6654 }

```

The horizontal rules

`\@@_draw_hrule:n` and the socket `draw-hrule` will appear in `\@@_draw_key_hlines:n` and in the token rules `\g_@@_rules_tl` (or within other functions used in `\g_@@_rules_tl`) and those token lists will be executed within a PGF pseudo-environment.

The argument `#1` is a list of *key=value* pairs.

```

6655 \cs_new_protected:Npn \@@_draw_hrule:n #1
6656 {
6657   {
6658     \int_set_eq:NN \l_@@_end_int \c@jCol
6659     \keys_set_known:nn { nicematrix / rules-after } { #1 }
6660     \socket_use:nn { nicematrix / draw-hrule }
6661   }
6662 }
```

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.

```

6663 \socket_new:nn { nicematrix / draw-hrule } { 0 }
6664 \socket_new_plug:nnn { nicematrix / draw-hrule } { general }
6665 {
6666   \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`.

```

6667   \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).

```

6668   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6669   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6670   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6671     \l_tmpb_tl
6672   {
```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```

6673     \@@_test_h:
6674     \bool_if:NTF \g_tmpa_bool
6675     {
6676       \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.

```

6677       { \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl }
6678       { \socket_use:n { nicematrix / draw-at-odd-vertex-h } }
6679     }
6680   {
6681     \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6682     {
6683       \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }
```

The socket `hsegment` is defined below with its four plugs.

```

6684         \socket_use:n { nicematrix / hsegment }
6685         \int_zero:N \l_@@_segment_start_int
6686     }
6687 }
6688 }
6689 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6690 {
6691     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6692     \socket_use:n { nicematrix / hsegment }
6693 }
6694 \group_end:
6695 }
6696 \socket_assign_plug:nn { nicematrix / draw-hrule } { general }
6697 \socket_new_plug:nnn { nicematrix / draw-hrule } { quick }
6698 {
6699     \group_begin:
6700     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6701     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6702     \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
6703     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6704     \socket_use:n { nicematrix / hsegment }
6705     \group_end:
6706 }

```

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.

```

6707 \cs_new_protected:Npn \@@_test_h:
6708 {
6709     \bool_gset_true:N \g_tmpa_bool
6710     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6711     \bool_if:NT \g_tmpa_bool
6712     {
6713         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6714         { \@@_test_hline_in_block:nnnnn ##1 }
6715         \bool_if:NT \g_tmpa_bool
6716         {
6717             \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6718             { \@@_test_hline_in_block:nnnnn ##1 }
6719             \bool_if:NT \g_tmpa_bool
6720             {
6721                 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6722                 { \@@_test_hline_in_stroken_block:nnnn ##1 }
6723             }
6724         }
6725     }
6726 }
6727 \socket_new:nn { nicematrix / draw-at-odd-vertex-h } { 0 }
6728 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-h } { active }
6729 {
6730     {
6731         \int_compare:nNnTF \l_@@_position_int > \c@iRow
6732         { \bool_set_false:N \l_tmpa_bool }
6733         {
6734             \@@_test_v:
6735             \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6736         }
6737         \int_compare:nNnTF \l_@@_position_int = 1
6738         { \bool_gset_false:N \g_tmpa_bool }

```

```

6739         {
6740             \tl_set:Nn \l_tmpa_tl { \int_eval:n { \l_tmpa_tl - 1 } }
6741             \@@_test_v:
6742         }
6743         \bool_gset:Nn \g_tmpa_bool
6744         { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6745     }
6746     \bool_if:NT \g_tmpa_bool
6747     {
6748         \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }
6749         \socket_use:n { nicematrix / hsegment }
6750         \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl
6751     }
6752 }

6753 \cs_new_protected:Npn \@@_test_in_corner_h:
6754 {
6755     \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6756     {
6757         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6758         { \bool_set_false:N \g_tmpa_bool }
6759     }
6760     {
6761         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6762         {
6763             \int_compare:nNnTF \l_tmpa_tl = 1
6764             { \bool_set_false:N \g_tmpa_bool }
6765             {
6766                 \@@_if_in_corner:nT
6767                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6768                 { \bool_set_false:N \g_tmpa_bool }
6769             }
6770         }
6771     }
6772 }

```

For the drawing of the (horizontal) segments, we will use a socket with four plugs :

- a plug `standar` for simple rules;
- a plug `multiplity` for the rules similar to the standard rules of `array` and `colortbl`, that is to say with multiplicity, etc;
- a plug `dotted` for the rules with our own system of dotted rules;
- a plug `tikz` for the rules drawn with TikZ.

```

6773 \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`.

```

6774 \socket_new_plug:nnn { nicematrix / hsegment } { standard }
6775 {
6776     \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6777     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6778     \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6779     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6780     \pgfusepathqstroke
6781 }
6782 \socket_assign_plug:nn { nicematrix / hsegment } { standard }

```

```

6783 \socket_new_plug:nnn { nicematrix / hsegment } { multiple }
6784 {
6785   \dim_add:Nn \l_@@_y_initial_dim
6786   {
6787     - 0.5 \l_@@_rule_width_after_dim
6788     +
6789     ( \arrayrulewidth * \l_@@_multiplicity_int
6790       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6791   }
6792   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6793   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6794   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6795   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6796   \bool_lazy_and:nnT
6797   { \cs_if_exist_p:N \CT@drsc@ }
6798   { ! \tl_if_blank_p:o \CT@drsc@ }
6799   {
6800     {
6801       \CT@drsc@
6802       \dim_set:Nn \l_@@_tmpd_dim
6803       {
6804         \l_@@_y_initial_dim - ( \doublerulesep + \arrayrulewidth )
6805         * ( \l_@@_multiplicity_int - 1 )
6806       }
6807       \pgfpathrectanglecorners
6808       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6809       { \pgfpoint \l_@@_x_final_dim \l_@@_tmpd_dim }
6810       \pgfusepathqfill
6811     }
6812   }
6813   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6814   \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
6815   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6816   {
6817     \dim_sub:Nn \l_@@_y_initial_dim { \arrayrulewidth + \doublerulesep }
6818     \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6819     \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
6820   }
6821   \pgfusepathqstroke
6822 }

```

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}
```

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{array}$$

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}
```

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{array}$$

```

6823 \socket_new_plug:nnn { nicematrix / hsegment } { dotted }
6824 {
6825   \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6826   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6827   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }

```

```

6828 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6829 \int_compare:nNnT \l_@@_segment_start_int = 1
6830 {
6831     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6832     \bool_if:NF \g_@@_delims_bool
6833     { \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.

```

6834     \tl_if_eq:NnF \g_@@_left_delim_tl (
6835         { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6836     )
6837     \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6838     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6839     \int_compare:nNnT \l_@@_segment_end_int = \c@jCol
6840     {
6841         \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6842         \bool_if:NF \g_@@_delims_bool
6843         { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6844         \tl_if_eq:NnF \g_@@_right_delim_tl )
6845         { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6846     }

```

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`

```

6847 \@@_draw_line:
6848 }

```

```

6849 \socket_new_plug:nnn { nicematrix / hsegment } { tikz }
6850 {
6851     {

```

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).

```

6852     \tl_if_empty:NF \l_@@_rule_color_tl
6853     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6854     \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6855     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6856     \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6857     \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6858     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6859     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6860     ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
6861     -- ( \l_@@_x_final_dim , \l_@@_y_initial_dim ) ;
6862 }
6863 }

```

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).

```

6864 \cs_new_protected:Npn \@@_draw_key_hlines:

```

```

6865 {
6866 {
6867   \dim_set_eq:Nn \l_@@_rule_width_after_dim \arrayrulewidth
6868   \int_set:Nn \l_@@_end_int \c@jCol

```

There is a currying in the following code.

```

6869   \str_if_eq:eeTF \l_@@_hlines_clist { all }
6870   { \@@_lines_step_inline:n \c@iRow }
6871   { \clist_map_inline:Nn \l_@@_hlines_clist }
6872   {
6873     \int_set:Nn \l_@@_position_int { ##1 }
6874     \socket_use:n { nicematrix / draw-hrule }
6875   }
6876 }
6877 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6878 \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.

```

6879 \cs_set:Npn \@@_Hline_i:n #1
6880 {
6881   \peek_remove_spaces:n
6882   {
6883     \peek_meaning:NTF \Hline
6884     { \@@_Hline_ii:nn { #1 + 1 } }
6885     { \@@_Hline_iii:n { #1 } }
6886   }
6887 }
6888 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6889 \cs_set:Npn \@@_Hline_iii:n #1
6890 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6891 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6892 {
6893   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6894   \skip_vertical:N \l_@@_rule_width_before_dim
6895   \tl_gput_right:Ne \g_@@_rules_tl
6896   {

```

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:

```

6897 {
6898   \int_compare:nNnT { #1 } > 1 { \@@_set_multiplicity:n { #1 } }
6899   \int_set:Nn \l_@@_position_int { \int_eval:n { \c@iRow + 1 } }
6900   \dim_set:Nn \l_@@_rule_width_after_dim
6901   { \dim_use:N \l_@@_rule_width_before_dim }
6902   \@@_draw_hrule:n { #2 }
6903 }
6904 }
6905 \egroup
6906 }

```

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`).

```

6907 \cs_new_protected:Npn \@@_custom_line:n #1
6908 {
6909   \str_clear:N \l_@@_letter_str
6910   \str_clear:N \l_@@_command_str
6911   \str_clear:N \l_@@_ccommand_str
6912   \tl_clear_new:N \l_@@_other_keys_tl
6913   \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
6914   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
6915   {
6916     \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }

```

We delete the last character which is a space.

```

6917     \str_set:Ne \l_@@_command_str
6918     { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
6919   }
6920   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
6921   {
6922     \str_set:Ne \l_@@_ccommand_str
6923     { \str_tail:N \l_@@_ccommand_str }
6924     \str_set:Ne \l_@@_ccommand_str
6925     { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
6926   }

```

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.

```

6927   \bool_lazy_all:nTF
6928   {
6929     { \str_if_empty_p:N \l_@@_letter_str }
6930     { \str_if_empty_p:N \l_@@_command_str }
6931     { \str_if_empty_p:N \l_@@_ccommand_str }
6932   }
6933   { \@@_error:n { No~letter~and~no~command } }
6934   { \@@_custom_line_i:o \l_@@_other_keys_tl }
6935 }
6936 \keys_define:nn { nicematrix / custom-line }
6937 {
6938   letter .str_set:N = \l_@@_letter_str ,
6939   letter .value_required:n = true ,
6940   command .str_set:N = \l_@@_command_str ,
6941   command .value_required:n = true ,
6942   ccommand .str_set:N = \l_@@_ccommand_str ,
6943   ccommand .value_required:n = true
6944 }

```

```

6945 \cs_new_protected:Npn \@@_custom_line_i:n #1
6946 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6947   \bool_set_false:N \l_@@_tikz_rule_bool
6948   \bool_set_false:N \l_@@_dotted_rule_bool
6949   \bool_set_false:N \l_@@_color_bool

```



```

6950 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6951 \bool_if:NT \l_@@_tikz_rule_bool
6952 {
6953   \IfPackageLoadedF { tikz }
6954   { \@@_error:n { tikz-in-custom-line-without-tikz } }
6955   \bool_if:NT \l_@@_color_bool
6956   { \@@_error:n { color-in-custom-line-with-tikz } }
6957 }
6958 \bool_if:NT \l_@@_dotted_rule_bool
6959 {
6960   \int_compare:nNnT \l_@@_multiplicity_int > 1
6961   { \@@_error:n { key-multiplicity-with-dotted } }
6962 }
6963 \str_if_empty:NF \l_@@_letter_str
6964 {
6965   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6966   { \@@_error:n { Several-letters } }
6967   {
6968     \tl_if_in:NoTF
6969     \c_@@_forbidden_letters_str
6970     \l_@@_letter_str
6971     { \@@_error:ne { Forbidden-letter } \l_@@_letter_str }
6972   }

```

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.

```

6973 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6974 { \@@_v_custom_line:nn { #1 } }
6975 }
6976 }
6977 }
6978 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6979 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6980 }
6981 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6982 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6983 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{nicematrix/rules-after}`). That's why the following set of keys has some keys which are no-op.

```

6984 \keys_define:nn { nicematrix / custom-line-bis }
6985 {
6986   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6987   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6988   color .value_required:n = true ,
6989   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6990   tikz .value_required:n = true ,
6991   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6992   dotted .value_forbidden:n = true ,
6993   total-width .code:n = { } ,
6994   total-width .value_required:n = true ,
6995   sep-color .code:n = { } ,
6996   sep-color .value_required:n = true ,
6997   unknown .code:n =
6998   \@@_unknown_key:nn
6999   { nicematrix / custom-line-bis }
7000   { Unknown-key-for-custom-line }
7001 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```
7002 \bool_new:N \l_@@_dotted_rule_bool
7003 \bool_new:N \l_@@_tikz_rule_bool
7004 \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).

```
7005 \keys_define:nn { nicematrix / custom-line-width }
7006 {
7007   multiplicity .int_set:N = \l_@@_tmpc_int ,
7008   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
7009   total-width .code:n = \dim_set:Nn \l_@@_rule_width_before_dim { #1 }
7010                       \bool_set_true:N \l_@@_total_width_bool ,
7011   total-width .value_required:n = true ,
7012   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7013 }
```

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_hrline:n` (which is in the internal `\CodeAfter`).

```
7014 \cs_new_protected:Npn \@@_h_custom_line:n #1
7015 {
```

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`).

```
7016   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
7017   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
7018 }
```

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_hrline:n` (which is in the internal `\CodeAfter`).

```
7019 \cs_new_protected:Npn \@@_c_custom_line:n #1
7020 {
```

Here, we need an expandable command since it begins with an `\noalign`.

```
7021   \exp_args:Nc \DeclareExpandableDocumentCommand
7022   { nicematrix - \l_@@_ccommand_str }
7023   { 0 { } m }
7024   {
7025     \noalign
7026     {
7027       \@@_compute_rule_width:n { #1 , ##1 }
7028       \skip_vertical:n \l_@@_rule_width_before_dim
7029       \clist_map_inline:nn
7030       { ##2 }
7031       { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
7032     }
7033   }
7034   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
7035 }
```

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*.

```
7036 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
7037 {
7038   \tl_if_in:nnTF { #2 } { - }
7039   { \@@_cut_on_hyphen:w #2 \q_stop }
7040   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
```

```

7041 \tl_gput_right:Ne \g_@@_rules_tl
7042 {
7043   \@@_draw_hrule:n
7044   {
7045     #1 ,
7046     start = \l_tmpa_tl ,
7047     end = \l_tmpb_tl ,
7048     position = \int_eval:n { \c@iRow + 1 } ,
7049     total-width = \dim_use:N \l_@@_rule_width_before_dim
7050   }
7051 }
7052 }
7053 \cs_new_protected:Npn \@@_compute_rule_width:n #1
7054 {
7055   \bool_set_false:N \l_@@_tikz_rule_bool
7056   \bool_set_false:N \l_@@_total_width_bool
7057   \bool_set_false:N \l_@@_dotted_rule_bool
7058   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
7059   \bool_if:NF \l_@@_total_width_bool
7060   {
7061     \bool_if:NTF \l_@@_dotted_rule_bool
7062     { \dim_set:Nn \l_@@_rule_width_before_dim { 2 \l_@@_xdots_radius_dim } }
7063     {
7064       \bool_if:NF \l_@@_tikz_rule_bool
7065       {
7066         \dim_set:Nn \l_@@_rule_width_before_dim
7067         {
7068           \arrayrulewidth * \l_@@_tmpc_int
7069           + \doublerulesep * ( \l_@@_tmpc_int - 1 )
7070         }
7071       }
7072     }
7073   }
7074 }

```

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

```

7075 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
7076 {
7077   \str_if_eq:nnTF { #2 } { [ ] }
7078   { \@@_v_custom_line_i:nw { #1 } [ ] }
7079   { \@@_v_custom_line_ii:nn { #2 } { #1 } }
7080 }
7081 \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
7082 { \@@_v_custom_line:nn { #1 , #2 } }
7083 \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
7084 {
7085   \@@_compute_rule_width:n { #2 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

7086 \tl_gput_right:Ne \g_@@_array_preamble_tl
7087 {
7088   \exp_not:N !
7089   { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_before_dim } }
7090 }
7091 \tl_gput_right:Ne \g_@@_rules_tl
7092 {

```

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.

```

7093     {
7094       \dim_set:Nn \l_@@_rule_width_after_dim
7095       { \dim_use:N \l_@@_rule_width_before_dim }
7096       \int_set:Nn \l_@@_position_int { \int_eval:n { \c@jCol + 1 } }
7097       \@@_draw_vrule:n { #2 }
7098     }
7099   }
7100   \@@_rec_preamble:n #1
7101 }
7102 \@@_custom_line:n
7103 { letter = : , command = \hdottedline , ccommand = \cdottedline, dotted }
7104 \AtBeginDocument
7105 {
7106   \IfPackageLoadedT { tikz }
7107   {
7108     \cs_if_exist:cTF { tikz@library@decorations@loaded }
7109     {
7110       \@@_custom_line:n
7111       {
7112         letter = ; ,
7113         command = \hdashedline ,
7114         ccommand = \cdashedline ,
7115         tikz =
7116         {
7117           dash~pattern = on~4~pt~off~2pt,
7118           dash~expand~off ,
7119           line~cap = butt
7120         } ,
7121         total-width = \pgflinewidth
7122       }
7123     }
7124     {
7125       \@@_custom_line:n
7126       {
7127         letter = ; ,
7128         command = \hdashedline ,
7129         ccommand = \cdashedline ,
7130         tikz = { dash~pattern = on~4~pt~off~2pt , line~cap = butt } ,
7131         total-width = \pgflinewidth
7132       }
7133     }
7134   }
7135 }

```

The key default-line

```

7136 \keys_define:nn { nicematrix / default-line }
7137 {
7138   multiplicity .int_set:N = \l_@@_multiplicity_int ,
7139   color .code:n = \bool_set_true:N \l_@@_color_bool ,
7140   color .value_required:n = true ,
7141   tikz .code:n =
7142     \bool_set_true:N \l_@@_tikz_rule_bool
7143     \tl_set:Nn \l_@@_tikz_rule_tl { #1 } ,
7144   tikz .value_required:n = true ,
7145   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7146   dotted .value_forbidden:n = true ,
7147   total-width .code:n = { } ,
7148   total-width .value_required:n = true ,

```

```

7149     sep-color .code:n = { } ,
7150     sep-color .value_required:n = true ,
7151     unknown .code:n =
7152         \@@_unknown_key:nn
7153         { nicematrix / default-line }
7154         { Unknown-key-for-default-line }
7155     }
7156 \cs_new_protected:Npn \@@_default_line:n #1
7157 {
7158     \bool_set_false:N \l_@@_tikz_rule_bool
7159     \bool_set_false:N \l_@@_dotted_rule_bool
7160     \bool_set_false:N \l_@@_color_bool
7161     \keys_set:nn { nicematrix / default-line } { #1 }
7162     \bool_if:NT \l_@@_tikz_rule_bool
7163     {
7164         \IfPackageLoadedF { tikz }
7165         { \@@_error:n { tikz-in-custom-line-without-tikz } }
7166         \bool_if:NT \l_@@_color_bool
7167         { \@@_error:n { color-in-custom-line-with-tikz } }
7168     }
7169     \bool_if:NT \l_@@_dotted_rule_bool
7170     {
7171         \int_compare:nNnT \l_@@_multiplicity_int > 1
7172         { \@@_error:n { key-multiplicity-with-dotted } }
7173     }
7174     \bool_if:NTF \l_@@_tikz_rule_bool
7175     {
7176         \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
7177         \socket_assign_plug:nn { nicematrix / hsegment } { tikz }
7178     }
7179     {
7180         \bool_if:NTF \l_@@_dotted_rule_bool
7181         {
7182             \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
7183             \socket_assign_plug:nn { nicematrix / hsegment } { dotted }
7184         }
7185         {
7186             \bool_if:NTF \l_@@_multiple_rule_bool
7187             {
7188                 \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
7189                 \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
7190             }
7191         }
7192     }
7193 }

```

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.

```

7194 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
7195 {
7196     \int_compare:nNnT \l_tmpa_tl > { #1 }
7197     {
7198         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
7199         {
7200             \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
7201             {
7202                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
7203                 { \bool_gset_false:N \g_tmpa_bool }
7204             }
7205         }
7206     }
7207 }

```

```

7206     }
7207 }

```

The same for vertical rules.

```

7208 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
7209 {
7210   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
7211   {
7212     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
7213     {
7214       \int_compare:nNnT \l_tmpb_tl > { #2 }
7215       {
7216         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
7217         { \bool_gset_false:N \g_tmpa_bool }
7218       }
7219     }
7220   }
7221 }

7222 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
7223 {
7224   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
7225   {
7226     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
7227     {
7228       \int_compare:nNnTF \l_tmpa_tl = { #1 }
7229       { \bool_gset_false:N \g_tmpa_bool }
7230       {
7231         \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
7232         { \bool_gset_false:N \g_tmpa_bool }
7233       }
7234     }
7235   }
7236 }

7237 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
7238 {
7239   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
7240   {
7241     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
7242     {
7243       \int_compare:nNnTF \l_tmpb_tl = { #2 }
7244       { \bool_gset_false:N \g_tmpa_bool }
7245       {
7246         \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
7247         { \bool_gset_false:N \g_tmpa_bool }
7248       }
7249     }
7250   }
7251 }

```

23 The empty corners

When the key `corners` is raised, the rules are not drawn in the corners; they are not colored and `\TikzEveryCell` does not apply. Of course, we have to compute the corners before we begin to draw the rules.

```

7252 \cs_new_protected:Npn \@@_compute_corners:
7253 {
7254   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7255   { \@@_mark_cells_of_block:nnnnn ##1 }

```

The list `\l_@@_corners_cells_clist` will be the list of all the empty cells (and not in a block) considered in the corners of the array. We use a `clist` instead of a `seq` because we will frequently search in that list (and searching in a `clist` is faster than searching in a `seq`).

```

7256 \clist_clear:N \l_@@_corners_cells_clist
7257 \clist_map_inline:Nn \l_@@_corners_cells_clist
7258 {
7259   \str_case:nnF { ##1 }
7260   {
7261     { NW }
7262     { \@@_compute_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
7263     { NE }
7264     { \@@_compute_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
7265     { SW }
7266     { \@@_compute_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
7267     { SE }
7268     { \@@_compute_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
7269   }
7270   { \@@_error:nn { bad~corner } { ##1 } }
7271 }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

7272 \clist_if_empty:NF \l_@@_corners_cells_clist
7273 {

```

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.

```

7274 \tl_gput_right:Ne \g_@@_aux_tl
7275 {
7276   \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
7277   { \l_@@_corners_cells_clist }
7278 }
7279 }
7280 }

```

```

7281 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
7282 {
7283   \int_step_inline:nnn { #1 } { #3 }
7284   {
7285     \int_step_inline:nnn { #2 } { #4 }
7286     { \cs_set_nopar:cpn { @@ _ block _ ##1 - #####1 } { } }
7287   }
7288 }

```

```

7289 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
7290 {
7291   \cs_if_exist:cTF
7292   { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
7293   \prg_return_true:
7294   \prg_return_false:
7295 }

```

“Computing a corner” is determining all the empty cells (which are not in a block) that belong to that corner. These cells will be added to the sequence `\l_@@_corners_cells_clist`.

The six arguments of `\@@_compute_corner:nnnnnn` are as follow:

- `#1` and `#2` are the number of row and column of the cell which is actually in the corner;
- `#3` and `#4` are the steps in rows and the step in columns when moving from the corner;
- `#5` is the number of the final row when scanning the rows from the corner;

- #6 is the number of the final column when scanning the columns from the corner.

```
7296 \cs_new_protected:Npn \@@_compute_corner:nnnnnn #1 #2 #3 #4 #5 #6
7297 {
```

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.

```
7298   \bool_set_false:N \l_tmpa_bool
7299   \int_zero_new:N \l_@@_last_empty_row_int
7300   \int_set:Nn \l_@@_last_empty_row_int { #1 }
7301   \int_step_inline:nnnn { #1 } { #3 } { #5 }
7302   {
7303     \bool_lazy_or:nnTF
7304     {
7305       \cs_if_exist_p:c
7306       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
7307     }
7308     { \@@_if_in_block_p:nn { ##1 } { #2 } }
7309     { \bool_set_true:N \l_tmpa_bool }
7310     {
7311       \bool_if:NF \l_tmpa_bool
7312       { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
7313     }
7314   }
```

Now, you determine the last empty cell in the row of number 1.

```
7315   \bool_set_false:N \l_tmpa_bool
7316   \int_zero_new:N \l_@@_last_empty_column_int
7317   \int_set:Nn \l_@@_last_empty_column_int { #2 }
7318   \int_step_inline:nnnn { #2 } { #4 } { #6 }
7319   {
7320     \bool_lazy_or:nnTF
7321     {
7322       \cs_if_exist_p:c
7323       { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
7324     }
7325     { \@@_if_in_block_p:nn { #1 } { ##1 } }
7326     { \bool_set_true:N \l_tmpa_bool }
7327     {
7328       \bool_if:NF \l_tmpa_bool
7329       { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
7330     }
7331   }
```

Now, we loop over the rows.

```
7332   \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
7333   {
```

We treat the row number `##1` with another loop.

```
7334     \bool_set_false:N \l_tmpa_bool
7335     \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
7336     {
7337       \bool_lazy_or:nnTF
7338       { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7339       { \@@_if_in_block_p:nn { ##1 } { #####1 } }
7340       { \bool_set_true:N \l_tmpa_bool }
7341       {
7342         \bool_if:NF \l_tmpa_bool
7343         {
7344           \int_set:Nn \l_@@_last_empty_column_int { #####1 }
7345           \clist_put_right:Nn
7346           \l_@@_corners_cells_clist
```



```

7347         { ##1 - #####1 }
7348         \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7349     }
7350 }
7351 }
7352 }
7353 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

7354 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
7355 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }

```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient:
`\clist_if_in:NcT \l_@@_corners_cells_clist { #1 } ...`

24 The environment {NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```

7356 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

7357 \keys_define:nn { nicematrix / NiceMatrixBlock }
7358 {
7359     auto-columns-width .code:n =
7360     {
7361         \bool_set_true:N \l_@@_block_auto_columns_width_bool
7362         \dim_gzero_new:N \g_@@_max_cell_width_dim
7363         \bool_set_true:N \l_@@_auto_columns_width_bool
7364     }
7365 }

7366 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
7367 {
7368     \int_gincr:N \g_@@_NiceMatrixBlock_int
7369     \dim_zero:N \l_@@_columns_width_dim
7370     \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
7371     \bool_if:NT \l_@@_block_auto_columns_width_bool
7372     {
7373         \cs_if_exist:cT
7374         { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
7375         {
7376             \dim_set:Nn \l_@@_columns_width_dim
7377             {
7378                 \use:c
7379                 { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7380             }
7381         }
7382     }
7383 }

```

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`).

```

7384 {
7385     \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.

```

7386     { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
7387     {
7388       \bool_if:NT \l_@@_block_auto_columns_width_bool
7389       {
7390         \iow_shipout:Nn \@mainaux \ExplSyntaxOn
7391         \iow_shipout:Ne \@mainaux
7392         {
7393           \cs_gset:cpn
7394           { @@ _ 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.

```

7395           { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7396         }
7397         \iow_shipout:Nn \@mainaux \ExplSyntaxOff
7398       }
7399     }
7400     \ignorespacesafterend
7401   }

```

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).

```

7402 \cs_new_protected:Npn \@@_create_extra_nodes:
7403 {
7404   \bool_if:nTF \l_@@_medium_nodes_bool
7405   {
7406     \bool_if:NTF \l_@@_no_cell_nodes_bool
7407     { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7408     {
7409       \bool_if:NTF \l_@@_large_nodes_bool
7410       \@@_create_medium_and_large_nodes:
7411       \@@_create_medium_nodes:
7412     }
7413   }
7414   {
7415     \bool_if:NT \l_@@_large_nodes_bool
7416     {
7417       \bool_if:NTF \l_@@_no_cell_nodes_bool
7418       { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7419       \@@_create_large_nodes:
7420     }
7421   }
7422 }

```

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 $\backslash c_max_dim$ or $-\backslash c_max_dim$.

```

7423 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7424 {
7425   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7426   {
7427     \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7428     \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7429     \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7430     \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7431   }
7432   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7433   {
7434     \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7435     \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7436     \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7437     \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7438   }

```

We begin the two nested loops over the rows and the columns of the array.

```

7439   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7440   {
7441     \int_step_variable:nnNn
7442     \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.

```

7443     {
7444       \cs_if_exist:cT
7445       { 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 $\backslash pgf@x$ and $\backslash pgf@y$.

```

7446     {
7447       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7448       \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
7449       { \dim_min:vn { l_@@_row_ \@@_i: _min_dim } \pgf@y }
7450       \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7451       {
7452         \dim_set:cn { l_@@_column_ \@@_j: _min_dim }
7453         { \dim_min:vn { l_@@_column_ \@@_j: _min_dim } \pgf@x }
7454       }

```

We retrieve the coordinates of the anchor north east of the (normal) node of the cell $(i-j)$. They will be stored in $\backslash pgf@x$ and $\backslash pgf@y$.

```

7455       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7456       \dim_set:cn { l_@@_row_ \@@_i: _max_dim }
7457       { \dim_max:vn { l_@@_row_ \@@_i: _max_dim } \pgf@y }
7458       \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7459       {
7460         \dim_set:cn { l_@@_column_ \@@_j: _max_dim }
7461         { \dim_max:vn { l_@@_column_ \@@_j: _max_dim } \pgf@x }
7462       }
7463     }
7464   }
7465 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7466 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7467 {
7468   \dim_compare:nnNt
7469   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
7470   {
7471     \@@_qpoint:n { row - \@@_i: - base }
7472     \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
7473     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
7474   }
7475 }
7476 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7477 {
7478   \dim_compare:nnNt
7479   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
7480   {
7481     \@@_qpoint:n { col - \@@_j: }
7482     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
7483     \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
7484   }
7485 }
7486 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7487 \cs_new_protected:Npn \@@_create_medium_nodes:
7488 {
7489   \pgfpicture
7490   \pgfrememberpicturepositiononpagetrue
7491   \pgf@relevantforpicturesizefalse
7492   \@@_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”.

```

7493   \tl_set:Nn \l_@@_suffix_tl { -medium }
7494   \@@_create_nodes:
7495   \endpgfpicture
7496 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones¹⁵. However, the computation of the mathematical coordinates of the “large nodes” needs the computation of the mathematical coordinates of the “medium nodes”. Hence, we use first `\@@_computations_for_medium_nodes:` and then the command `\@@_computations_for_large_nodes:`.

```

7497 \cs_new_protected:Npn \@@_create_large_nodes:
7498 {
7499   \pgfpicture
7500   \pgfrememberpicturepositiononpagetrue
7501   \pgf@relevantforpicturesizefalse
7502   \@@_computations_for_medium_nodes:
7503   \@@_computations_for_large_nodes:
7504   \tl_set:Nn \l_@@_suffix_tl { - large }
7505   \@@_create_nodes:
7506   \endpgfpicture
7507 }
7508 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7509 {

```

¹⁵If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7510 \pgfpicture
7511 \pgfrememberpicturepositiononpagetrue
7512 \pgf@relevantforpicturesizefalse
7513 \@@_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”.

```

7514 \tl_set:Nn \l_@@_suffix_tl { - medium }
7515 \@@_create_nodes:
7516 \@@_computations_for_large_nodes:
7517 \tl_set:Nn \l_@@_suffix_tl { - large }
7518 \@@_create_nodes:
7519 \endpgfpicture
7520 }

```

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.

```

7521 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7522 {
7523 \int_set:Nn \l_@@_first_row_int 1
7524 \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`.

```

7525 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7526 {
7527 \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7528 {
7529 (
7530 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7531 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7532 )
7533 / 2
7534 }
7535 \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7536 { l_@@_row _ \@@_i: _ min_dim }
7537 }
7538 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7539 {
7540 \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7541 {
7542 (
7543 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7544 \dim_use:c
7545 { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7546 )
7547 / 2
7548 }
7549 \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7550 { l_@@_column _ \@@_j: _ max _ dim }
7551 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7552 \dim_sub:cn
7553 { l_@@_column _ 1 _ min _ dim }
7554 \l_@@_left_margin_dim
7555 \dim_add:cn
7556 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7557 \l_@@_right_margin_dim
7558 }

```

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).

```

7559 \cs_new_protected:Npn \@@_create_nodes:
7560 {
7561   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7562   {
7563     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7564     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7565       \@@_pgf_rect_node:nnnnn
7566       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7567       { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7568       { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7569       { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
7570       { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7571       \str_if_empty:NF \l_@@_name_str
7572       {
7573         \pgfnodealias
7574         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7575         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7576       }
7577     }
7578   }
7579   \int_step_inline:nn \c@iRow
7580   {
7581     \pgfnodealias
7582     { \@@_env: - ##1 - last \l_@@_suffix_tl }
7583     { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7584   }
7585   \int_step_inline:nn \c@jCol
7586   {
7587     \pgfnodealias
7588     { \@@_env: - last - ##1 \l_@@_suffix_tl }
7589     { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7590   }
7591   \pgfnodealias
7592   { \@@_env: - last - last \l_@@_suffix_tl }
7593   { \@@_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 .

```

7594   \seq_map_pairwise_function:NNN
7595   \g_@@_multicolumn_cells_seq
7596   \g_@@_multicolumn_sizes_seq
7597   \@@_node_for_multicolumn:nn
7598 }

```

```

7599 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7600 {
7601   \cs_set_nopar:Npn \@@_i: { #1 }
7602   \cs_set_nopar:Npn \@@_j: { #2 }
7603 }

```

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”).

```

7604 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7605 {
7606   \@@_extract_coords_values: #1 \q_stop

```

```

7607 \@@_pgf_rect_node:nnnnn
7608 { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7609 { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7610 { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7611 { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7612 { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7613 \str_if_empty:NF \l_@@_name_str
7614 {
7615   \pgfnodealias
7616   { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7617   { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7618 }
7619 }

```

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).

```

7620 \keys_define:nn { nicematrix / BlockFirstPass }
7621 {
7622   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7623   \bool_set_true:N \l_@@_p_block_bool ,
7624   j .value_forbidden:n = true ,
7625   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7626   l .value_forbidden:n = true ,
7627   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7628   r .value_forbidden:n = true ,
7629   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7630   c .value_forbidden:n = true ,
7631   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7632   L .value_forbidden:n = true ,
7633   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7634   R .value_forbidden:n = true ,
7635   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7636   C .value_forbidden:n = true ,
7637   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7638   t .value_forbidden:n = true ,
7639   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7640   T .value_forbidden:n = true ,
7641   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7642   b .value_forbidden:n = true ,
7643   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7644   B .value_forbidden:n = true ,
7645   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7646   m .value_forbidden:n = true ,
7647   v-center .meta:n = m ,
7648   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7649   p .value_forbidden:n = true ,
7650   respect-arraystretch .code:n =
7651   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7652   respect-arraystretch .value_forbidden:n = true ,
7653   color .code:n =
7654   \@@_color:n { #1 }
7655   \tl_set_rescan:Nnn
7656   \l_@@_draw_tl

```

```

7657     { \char_set_catcode_other:N ! }
7658     { #1 } ,
7659     color .value_required:n = true ,
7660 }

```

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.

```

7661 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7662 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7663 {

```

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).

```

7664     \tl_if_blank:nTF { #2 }
7665     { \@@_Block_ii:nnnnn 1 1 }
7666     {
7667         \tl_if_in:nnTF { #2 } { - }
7668         {
7669             \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7670             \@@_Block_i_czech:w \@@_Block_i:w
7671             #2 \q_stop
7672         }
7673         {
7674             \@@_error:nn { Bad-argument~for~Block } { #2 }
7675             \@@_Block_ii:nnnnn 1 1
7676         }
7677     }
7678     { #1 } { #3 } { #4 }
7679     \ignorespaces
7680 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7681 \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`.

```

7682 {
7683     \char_set_catcode_active:N -
7684     \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7685 }

```

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.

```

7686 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7687 {

```

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).

```

7688     \bool_lazy_or:nnTF
7689     { \tl_if_blank_p:n { #1 } }
7690     { \str_if_eq_p:ee { * } { #1 } }
7691     { \int_set:Nn \l_tmpa_int { 100 } }
7692     { \int_set:Nn \l_tmpa_int { #1 } }
7693     \bool_lazy_or:nnTF

```



```

7694 { \tl_if_blank_p:n { #2 } }
7695 { \str_if_eq_p:ee { * } { #2 } }
7696 { \int_set:Nn \l_tmpb_int { 100 } }
7697 { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7698 \int_compare:nNnTF \l_tmpb_int = 1
7699 {
7700   \tl_if_empty:NNTF \l_@@_hpos_cell_tl
7701   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7702   { \str_set:Nn \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7703 }
7704 { \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.

```

7705 \keys_set_known:n { nicematrix / BlockFirstPass } { #3 }
7706 \tl_set:Nn \l_tmpa_tl
7707 {
7708   { \int_use:N \c@iRow }
7709   { \int_use:N \c@jCol }
7710   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7711   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7712 }

```

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).

```

7713 \bool_set_false:N \l_tmpa_bool
7714 \bool_if:NT \l_@@_amp_in_blocks_bool

```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

7715 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7716 \bool_case:nF
7717 {
7718   \l_tmpa_bool { \@@_Block_vii:eennn }
7719   \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.

```

7720 \l_@@_X_bool { \@@_Block_v:eennn }
7721 { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eennn }
7722 { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7723 { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7724 }
7725 { \@@_Block_v:eennn }
7726 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7727 }

```

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.

```

7728 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7729 {
7730   \int_gincr:N \g_@@_block_box_int
7731   \cs_set_eq:NN \cellcolor \@@_cellcolor_error
7732   \cs_set_eq:NN \rowcolor \@@_rowcolor_error
7733   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7734   {
7735     \tl_gput_right:Ne \g_@@_rules_tl
7736     {
7737       \@@_draw_diagbox:nnnnnn
7738       { \int_use:N \c@iRow }
7739       { \int_use:N \c@jCol }
7740       { \int_eval:n { \c@iRow + #1 - 1 } }
7741       { \int_eval:n { \c@jCol + #2 - 1 } }
7742       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7743       { \g_@@_row_style_tl \exp_not:n { ##2 } }
7744     }
7745   }
7746   \box_gclear_new:c
7747   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful*: if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7748 \hbox_gset:cn
7749 { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
7750 {

```

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`).

```

7751 \tl_if_empty:NTF \l_@@_color_tl
7752 { \int_compare:nNnT { #2 } = 1 \set@color }
7753 { \@@_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`.

```

7754 \int_compare:nNnT { #1 } = 1
7755 {
7756   \int_if_zero:nTF \c@iRow
7757   {

```

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}$

```

```

7758      \cs_set_eq:NN \Block \@@_NullBlock:
7759      \l_@@_code_for_first_row_tl
7760    }
7761    {
7762      \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7763      {
7764        \cs_set_eq:NN \Block \@@_NullBlock:
7765        \l_@@_code_for_last_row_tl
7766      }
7767    }
7768    \g_@@_row_style_tl
7769  }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7770      \@@_reset_arraystretch:
7771      \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7772      #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.).

```

7773      \@@_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}`.

```

7774      \bool_if:NTF \l_@@_tabular_bool
7775      {
7776        \bool_lazy_all:nTF
7777        {
7778          { \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.

```

7779          { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7780          { ! \g_@@_rotate_bool }
7781        }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7782      {
7783        \use:e
7784        {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7785      \exp_not:N \begin { minipage }
7786      [ \str_lowercase:f \l_@@_vpos_block_str ]
7787      { \l_@@_col_width_dim }
7788      \str_case:on \l_@@_hpos_block_str
7789      { c \centering r \raggedleft l \raggedright }
7790    }
7791    #5
7792  \end { minipage }
7793  }

```

In the other cases, we use a `{tabular}`.

```

7794         {
7795             \use:e
7796             {
Curiously, \exp_not:N is still mandatory when tagging=on.
7797             \exp_not:N \begin { tabular }
7798             [ \str_lowercase:f \l_@@_vpos_block_str ]
7799             { @ { } \l_@@_hpos_block_str @ { } }
7800             }
7801             #5
7802             \end { tabular }
7803         }
7804     }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is `false`). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7805     {
7806         $ % $
7807         \bool_if:NT \l_@@_small_bool % 2026/04/05
7808         {
7809             \def \arraystretch { 0.47 }
7810             \dim_set:Nn \arraycolsep { 1.45 pt }
7811         }
7812         \use:e
7813         {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7814             \exp_not:N \begin { array }
7815             [ \str_lowercase:f \l_@@_vpos_block_str ]
7816             { @ { } \l_@@_hpos_block_str @ { } }
7817         }
7818         \@@_tuning_key_small: % 2026/04/05
7819         #5
7820         \end { array }
7821         $ % $
7822     }
7823 }

```

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).

```

7824     \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.

```

7825     \int_compare:nNnT { #2 } = 1
7826     {
7827         \dim_gset:Nn \g_@@_blocks_wd_dim
7828         {
7829             \dim_max:nn
7830             \g_@@_blocks_wd_dim
7831             {
7832                 \box_wd:c
7833                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7834             }
7835         }
7836     }

```

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 explicetely 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.

```

7837     \int_compare:nNnT { #1 } = 1

```

```

7838 {
7839     \bool_lazy_any:nT
7840     {
7841         { \str_if_empty_p:N \l_@@_vpos_block_str }
7842         { \str_if_eq_p:ee t \l_@@_vpos_block_str }
7843         { \str_if_eq_p:ee b \l_@@_vpos_block_str }
7844     }
7845     \@@_adjust_blocks_ht_dp:
7846 }
7847 \seq_gput_right:Ne \g_@@_blocks_seq
7848 {
7849     \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.

```

7850 {
7851     \exp_not:n { #3 } ,
7852     \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7853     \bool_if:NT \g_@@_rotate_bool
7854     {
7855         \bool_if:NTF \g_@@_rotate_c_bool
7856         { m }
7857         {
7858             \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7859             { T }
7860         }
7861     }
7862 }
7863 {
7864     \box_use_drop:c
7865     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7866 }
7867 }
7868 \bool_set_false:N \g_@@_rotate_c_bool
7869 }

7870 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7871 {
7872     \dim_gset:Nn \g_@@_blocks_ht_dim
7873     {
7874         \dim_max:nn
7875         \g_@@_blocks_ht_dim
7876         {
7877             \box_ht:c
7878             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7879         }
7880     }
7881     \dim_gset:Nn \g_@@_blocks_dp_dim
7882     {
7883         \dim_max:nn
7884         \g_@@_blocks_dp_dim
7885         {
7886             \box_dp:c
7887             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7888         }
7889     }
7890 }

```

```

7891 \cs_new:Npn \@@_adjust_hpos_rotate:
7892 {

```

```

7893 \bool_if:NT \g_@@_rotate_bool
7894 {
7895   \str_set:Ne \l_@@_hpos_block_str
7896   {
7897     \bool_if:NTF \g_@@_rotate_c_bool
7898     c
7899     {
7900       \str_case:onF \l_@@_vpos_block_str
7901       { b l B l t r T r }
7902       {
7903         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
7904         r
7905         l
7906       }
7907     }
7908   }
7909 }
7910 }
7911 \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*.

```

7912 \cs_new_protected:Npn \@@_rotate_box_of_block:
7913 {
7914   \box_grotate:cn
7915   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7916   { \bool_if:NTF \g_@@_rotate_minus_bool { -90 } { 90 } }
7917   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7918   {
7919     \vbox_gset_top:cn
7920     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7921     {
7922       \skip_vertical:n { 0.8 ex }
7923       \box_use:c
7924       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7925     }
7926   }
7927   \bool_if:NT \g_@@_rotate_c_bool
7928   {
7929     \hbox_gset:cn
7930     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7931     {
7932       $ % $
7933       \vcenter
7934       {
7935         \box_use:c
7936         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7937       }
7938       $ % $
7939     }
7940   }
7941 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key p). In that case, the content of the block is *not* composed right away in a box. The composition in a box will be done further, just after the construction of the array (cf. \@@_draw_blocks: and above all \@@_Block_v:nnnnn).

#1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of key=values pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7942 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7943 {

```

```

7944 \seq_gput_right:Ne \g_@@_blocks_seq
7945 {
7946   \l_tmpa_tl
7947   { \exp_not:n { #3 } }
7948   {
7949     \bool_if:NTF \l_@@_tabular_bool
7950     {
7951       {

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7952 \@@_reset_arraystretch:
7953 \exp_not:n
7954 {
7955   \dim_zero:N \extrarowheight
7956   #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).

```

7957 \tag_if_active:T { \tag_stop:n { table } }
7958 \use:e
7959 {
7960   \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7961   { @ { } \l_@@_hpos_block_str @ { } }
7962   }
7963   #5
7964 \end { tabular }
7965 }
7966 }
7967 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7968 {
7969   {

```

The following will be no-op when `respect-arraystretch` is in force.

```

7970 \@@_reset_arraystretch:
7971 \exp_not:n
7972 {
7973   \dim_zero:N \extrarowheight
7974   #4
7975   $ % $
7976   \use:e
7977   {
7978     \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7979     { @ { } \l_@@_hpos_block_str @ { } }
7980   }
7981   \@@_tuning_key_small: % 2026/05/04
7982   #5
7983   \end { array }
7984   $ % $
7985   }
7986   }
7987 }
7988 }
7989 }
7990 }
7991 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7992 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7993 {

```

```

7994 \seq_gput_right:Ne \g_@@_blocks_seq
7995 {
7996   \l_tmpa_tl
7997   { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7998   { { \exp_not:n { #4 #5 } } }
7999 }
8000 }
8001 \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`.

```

8002 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
8003 {
8004   \seq_gput_right:Ne \g_@@_blocks_seq
8005   {
8006     \l_tmpa_tl
8007     { \exp_not:n { #3 } }
8008     { \exp_not:n { #4 #5 } }
8009   }
8010 }
8011 \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).

```

8012 \keys_define:nn { nicematrix / Block }
8013 {
8014   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
8015   &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

8016 tikz .code:n =
8017   \IfPackageLoadedTF { tikz }
8018   { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
8019   { \@@_error:n { tikz~key~without~tikz } } ,
8020 tikz .value_required:n = true ,
8021 fill .code:n =
8022   \tl_set_rescan:Nnn
8023   \l_@@_fill_tl
8024   { \char_set_catcode_other:N ! }
8025   { #1 } ,
8026 fill .value_required:n = true ,

```

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

```

8027 opacity .tl_set:N = \l_@@_opacity_tl ,
8028 opacity .value_required:n = true ,
8029 draw .code:n =
8030   \tl_set_rescan:Nnn
8031   \l_@@_draw_tl
8032   { \char_set_catcode_other:N ! }
8033   { #1 } ,
8034 draw .default:n = default ,
8035 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8036 rounded-corners .default:n = 4 pt ,
8037 color .code:n =
8038   \@@_color:n { #1 }
8039   \tl_set_rescan:Nnn
8040   \l_@@_draw_tl
8041   { \char_set_catcode_other:N ! }
8042   { #1 } ,
8043 borders .clist_set:N = \l_@@_borders_clist ,

```



```

8044     borders .value_required:n = true ,
8045     hvlines .meta:n = { vlines , hlines } ,
8046     vlines .bool_set:N = \l_@@_vlines_block_bool ,
8047     hlines .bool_set:N = \l_@@_hlines_block_bool ,
8048     rules/width .dim_set:N = \arrayrulewidth ,
8049     rules/color .tl_set:N = \l_@@_rules_color_tl ,
8050     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,

```

The key `line-width` is now deprecated (replaced by `rules/width`).

```

8051     line-width .dim_set:N = \arrayrulewidth ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `BlockFirstPass`.

```

8052     j .code:n = \str_set:Nn \l_@@_hpos_block_str j
8053               \bool_set_true:N \l_@@_p_block_bool ,
8054     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
8055     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
8056     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
8057     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
8058               \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8059     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
8060               \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8061     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
8062               \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8063     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
8064     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
8065     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
8066     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
8067     m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
8068     m .value_forbidden:n = true ,
8069     v-center .meta:n = m ,
8070     p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
8071     p .value_forbidden:n = true ,
8072     name .str_set:N = \l_@@_block_name_str ,
8073     name .value_required:n = true ,
8074     respect-arraystretch .code:n =
8075       \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
8076     respect-arraystretch .value_forbidden:n = true ,
8077     transparent .bool_set:N = \l_@@_transparent_bool ,
8078     unknown .code:n =
8079       \@@_unknown_key:nn
8080       { nicematrix / Block }
8081       { Unknown-key-for-Block }
8082   }

```

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.

```

8083 \cs_new_protected:Npn \@@_draw_blocks:
8084 {
8085   \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
8086   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
8087 }
8088 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
8089 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

8090   \int_zero:N \l_@@_last_row_int
8091   \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”).

```

8092 \int_compare:nNnTF { #3 } > { 98 }
8093 { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
8094 { \int_set:Nn \l_@@_last_row_int { #3 } }
8095 \int_compare:nNnTF { #4 } > { 98 }
8096 { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
8097 { \int_set:Nn \l_@@_last_col_int { #4 } }
8098 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
8099 {
8100   \bool_lazy_and:nnTF
8101     \l_@@_preamble_bool
8102     {
8103       \int_compare_p:n
8104         { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
8105     }
8106     {
8107       \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
8108       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
8109       \@@_msg_redirect_name:nn { columns-not-used } { none }
8110     }
8111     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8112   }
8113   {
8114     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
8115     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8116     {

```

We expand the four first arguments of \@@_Block_v:nnnnnn.

```

8117 \use:e
8118 {
8119   \@@_Block_v:nnnnnn
8120   { #1 }
8121   { #2 }
8122   { \int_use:N \l_@@_last_row_int }
8123   { \int_use:N \l_@@_last_col_int }
8124 }
8125 { #5 }
8126 { #6 }
8127 }
8128 }
8129 }

```

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.

```

8130 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
8131 {

```

The group is for the keys.

```

8132 \group_begin:
8133 \int_compare:nNnT { #1 } = { #3 }
8134 { \str_set:Nn \l_@@_vpos_block_str { t } }
8135 \keys_set:nn { nicematrix / Block } { #5 }

```

If the content of the block contains &, we will have a special treatment (since the cell must be divided in several sub-cells). Remark that \tl_if_in:nnT is faster then \str_if_in:nnT.

```

8136 \bool_if:NT \l_@@_amp_in_blocks_bool
8137 { \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool } }
8138 \bool_lazy_and:nnT
8139   \l_@@_vlines_block_bool
8140   { ! \l_@@_ampersand_bool }
8141 {

```

```

8142     \tl_gput_right:Ne \g_@@_rules_tl
8143     {
8144         \@@_vlines_block:nnnnnn
8145         { \dim_use:N \arrayrulewidth }
8146         { \l_@@_rules_color_tl }
8147         { #1 } { #2 } { #3 } { #4 }
8148     }
8149 }
8150 \bool_if:NT \l_@@_hlines_block_bool
8151 {
8152     \tl_gput_right:Ne \g_@@_rules_tl
8153     {
8154         \@@_hlines_block:nnnnnn
8155         { \dim_use:N \arrayrulewidth }
8156         { \l_@@_rules_color_tl }
8157         { #1 } { #2 } { #3 } { #4 }
8158     }
8159 }

```

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).

```

8160 \bool_if:NF \l_@@_transparent_bool
8161 {
8162     \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8163     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
8164 }

8165 \tl_if_empty:NF \l_@@_draw_tl
8166 {
8167     \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
8168     { \@@_error:n { hlines-with-color } }
8169     \tl_gput_right:Nn \g_@@_rules_tl
8170     {
8171         \@@_stroke_block:nnnnn

```

#5 are the options

```

8172         { #5 } { #1 } { #2 } { #3 } { #4 }
8173     }
8174     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
8175     { { #1 } { #2 } { #3 } { #4 } }
8176 }

8177 \clist_if_empty:NF \l_@@_borders_clist
8178 {
8179     \tl_gput_right:Nn \g_nicematrix_code_after_tl
8180     {
8181         \@@_stroke_borders_block:nnnnn
8182         { #5 } { #1 } { #2 } { #3 } { #4 }
8183     }
8184 }

8185 \tl_if_empty:NF \l_@@_fill_tl
8186 {
8187     \@@_add_opacity_to_fill:
8188     \tl_gput_right:Ne \g_@@_pre_code_before_tl
8189     {
8190         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
8191         { #1 - #2 }
8192         { #3 - #4 }
8193         { \dim_use:N \l_@@_rounded_corners_dim }
8194     }
8195 }

```

```

8196 \seq_if_empty:NF \l_@@_tikz_seq
8197 {
8198   \tl_gput_right:Ne \g_nicematrix_code_before_tl
8199   {
8200     \@@_block_tikz:nnnnn
8201     { \seq_use:Nn \l_@@_tikz_seq { , } }
8202     { #1 } { #2 } { #3 } { #4 }

```

We will have in that last field a list of lists of TikZ keys.

```

8203   }
8204 }

8205 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
8206 {
8207   \tl_gput_right:Ne \g_@@_rules_tl
8208   {
8209     \@@_draw_diagbox:nnnnnn
8210     { #1 } { #2 } { #3 } { #4 }
8211     { \exp_not:n { ##1 } }
8212     { \exp_not:n { ##2 } }
8213   }
8214 }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} & & one & \\
& & two & \\
three & & four & five & \\
six & & seven & eight & \\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block		one
		two
three	four	five
six	seven	eight

We highlight the node `1-1-block-short`

our block		one
		two
three	four	five
six	seven	eight

The construction of the node corresponding to the merged cells.

```

8215 \pgfpicture
8216 \pgfrememberpicturepositiononpagetrue
8217 \pgf@relevantforpicturesizefalse
8218 \@@_qpoint:n { row - #1 }
8219 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8220 \@@_qpoint:n { col - #2 }
8221 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8222 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
8223 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8224 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8225 \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.

```

8226 \@@_pgf_rect_node:nnnnn
8227 { \@@_env: - #1 - #2 - block }
8228 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8229 \str_if_empty:NF \l_@@_block_name_str
8230 {

```

```

8231 \pgfnodealias
8232 { \l_@@_env: - \l_@@_block_name_str }
8233 { \l_@@_env: - #1 - #2 - block }
8234 \str_if_empty:NF \l_@@_name_str
8235 {
8236 \pgfnodealias
8237 { \l_@@_name_str - \l_@@_block_name_str }
8238 { \l_@@_env: - #1 - #2 - block }
8239 }
8240 }

```

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.

```

8241 \bool_if:NF \l_@@_hpos_of_block_cap_bool
8242 {
8243 \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.

```

8244 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8245 {

```

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.

```

8246 \cs_if_exist:cT
8247 { pgf @ sh @ ns @ \l_@@_env: - ##1 - #2 }
8248 {
8249 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8250 {
8251 \pgfpointanchor { \l_@@_env: - ##1 - #2 } { west }
8252 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
8253 }
8254 }
8255 }

```

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.

```

8256 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
8257 {
8258 \l_@@_qpoint:n { col - #2 }
8259 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8260 }
8261 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
8262 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8263 {
8264 \cs_if_exist:cT
8265 { pgf @ sh @ ns @ \l_@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8266 {
8267 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8268 {
8269 \pgfpointanchor
8270 { \l_@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8271 { east }
8272 \dim_set:Nn \l_@@_tmpd_dim
8273 { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
8274 }
8275 }
8276 }
8277 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
8278 {
8279 \l_@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8280 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x

```

```

8281     }
8282     \@_pgf_rect_node:nnnnn
8283     { \@_env: - #1 - #2 - block - short }
8284     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8285 }

```

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.

```

8286 \bool_if:NT \l_@@_medium_nodes_bool
8287 {
8288     \@_pgf_rect_node:nnn
8289     { \@_env: - #1 - #2 - block - medium }
8290     { \pgfpointanchor { \@_env: - #1 - #2 - medium } { north-west } }
8291     {
8292         \pgfpointanchor
8293         { \@_env:
8294             - \int_use:N \l_@@_last_row_int
8295             - \int_use:N \l_@@_last_col_int - medium
8296         }
8297         { south-east }
8298     }
8299 }
8300 \endpgfpicture
8301

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand &.

```

8302 \bool_if:NTF \l_@@_ampersand_bool
8303 {
8304     \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
8305     \int_zero_new:N \l_@@_split_int
8306     \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).

```

8307     \int_set:Nn \l_@@_first_row_int { #1 }
8308     \int_set:Nn \l_@@_first_col_int { #2 }
8309     \int_set:Nn \l_@@_last_row_int { #3 }
8310     \int_set:Nn \l_@@_last_col_int { #4 }
8311
8312     \pgfrememberpicturepositiononpagetrue
8313     \pgf@relevantforpicturesizefalse
8314     \@_qpoint:n { row - #1 }
8315     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8316     \@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8317     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
8318     \@_qpoint:n { col - #2 }
8319     \dim_set_eq:NN \l_tmpa_dim \pgf@x
8320     \@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8321     \dim_set:Nn \l_tmpb_dim
8322     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
8323     \bool_lazy_or:nnT
8324     \l_@@_vlines_block_bool
8325     { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
8326     {
8327         \int_step_inline:nn { \l_@@_split_int - 1 }
8328         {
8329             \pgfpathmoveto
8330             {
8331                 \pgfpoint
8332                 { \l_tmpa_dim + ##1 \l_tmpb_dim }
8333                 \l_@@_tmpc_dim
8334             }

```

```

8335         \pgfpathlineto
8336         {
8337             \pgfpoint
8338             { \l_tmpa_dim + ##1 \l_tmpb_dim }
8339             \l_@@_tmpd_dim
8340         }
8341         \CT@arc@
8342         \pgfsetlinewidth { 1.1 \arrayrulewidth }
8343         \pgfsetrectcap
8344         \pgfusepathqstroke
8345     }
8346 }
8347 \cs_set_eq:NN \cellcolor \@@_subcellcolor
8348 \int_zero_new:N \l_@@_split_i_int
8349 \str_if_eq:eeTF \l_@@_vpos_block_str T
8350 {
8351     \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
8352     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8353 }
8354 {
8355     \str_if_eq:eeTF \l_@@_vpos_block_str B
8356     {
8357         \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
8358         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8359     }
8360     {
8361         \bool_lazy_or:nnTF
8362         { \int_compare_p:nNn { #1 } = { #3 } }
8363         { \str_if_eq_p:ee \l_@@_vpos_block_str t }
8364         {
8365             \@@_qpoint:n { row - #1 - base }
8366             \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8367         }
8368         {
8369             \str_if_eq:eeTF \l_@@_vpos_block_str b
8370             {
8371                 \@@_qpoint:n { row - #3 - base }
8372                 \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8373             }
8374             {
8375                 \@@_qpoint:n { #1 - #2 - block }
8376                 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8377             }
8378         }
8379     }
8380 }
8381 \int_step_inline:nn \l_@@_split_int
8382 {
8383     \group_begin:
The counter \l_@@_split_i_int is only for the command \@@_subcellcolor.
8384     \int_set:Nn \l_@@_split_i_int { ##1 }
8385     \dim_set:Nn \col@sep
8386     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
8387     \pgftransformshift
8388     {
8389         \pgfpoint
8390         {
8391             \l_tmpa_dim + ##1 \l_tmpb_dim -
8392             \str_case:on \l_@@_hpos_block_str
8393             {
8394                 l { \l_tmpb_dim + \col@sep }
8395                 c { 0.5 \l_tmpb_dim }
8396                 r { \col@sep }

```

```

8397         }
8398     }
8399     { \l_@@_tmpc_dim }
8400 }
8401 \pgfset { inner~sep = \c_zero_dim }
8402 \pgfnode
8403 { rectangle }
8404 {
8405     \str_if_eq:eeTF T \l_@@_vpos_block_str
8406     {
8407         \str_case:on \l_@@_hpos_block_str
8408         {
8409             l { north~west }
8410             c { north }
8411             r { north~east }
8412         }
8413     }
8414     {
8415         \str_if_eq:eeTF B \l_@@_vpos_block_str
8416         {
8417             \str_case:on \l_@@_hpos_block_str
8418             {
8419                 l { south~west }
8420                 c { south }
8421                 r { south~east }
8422             }
8423         }
8424         {
8425             \bool_lazy_any:nTF
8426             {
8427                 { \int_compare_p:nNn { #1 } = { #3 } }
8428                 { \str_if_eq_p:ee t \l_@@_vpos_block_str }
8429                 { \str_if_eq_p:ee b \l_@@_vpos_block_str }
8430             }
8431             {
8432                 \str_case:on \l_@@_hpos_block_str
8433                 {
8434                     l { base~west }
8435                     c { base }
8436                     r { base~east }
8437                 }
8438             }
8439             {
8440                 \str_case:on \l_@@_hpos_block_str
8441                 {
8442                     l { west }
8443                     c { center }
8444                     r { east }
8445                 }
8446             }
8447         }
8448     }
8449 }
8450 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8451 \group_end:
8452 }
8453 \endpgfpicture
8454 }

```

Now the case where there is no ampersand & in the content of the block.

```

8455 {
8456     \bool_if:NTF \l_@@_p_block_bool
8457     {

```


When the final user has used the key `p`, we have to compute the width.

```

8458 \pgfpicture
8459 \pgfrememberpicturepositiononpagetrue
8460 \pgf@relevantforpicturesizefalse
8461 \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8462 {
8463   \@@_qpoint:n { col - #2 }
8464   \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8465   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8466 }
8467 {
8468   \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8469   \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8470   \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8471 }
8472 \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8473 \endpgfpicture
8474 \hbox_set:Nn \l_@@_cell_box
8475 {
8476   \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8477     { \g_tmpb_dim }
8478     \str_case:on \l_@@_hpos_block_str
8479       { c \centering r \raggedleft l \raggedright j { } }
8480     #6
8481   \end { minipage }
8482 }
8483 }
8484 {
8485   \hbox_set:Nn \l_@@_cell_box
8486   {
8487     \set@color
8488     #6
8489   }
8490 }
8491 \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.

```

8492 \pgfpicture
8493 \pgfrememberpicturepositiononpagetrue
8494 \pgf@relevantforpicturesizefalse
8495 \bool_lazy_any:nTF
8496 {
8497   { \str_if_empty_p:N \l_@@_vpos_block_str }
8498   { \str_if_eq_p:ee c \l_@@_vpos_block_str }
8499   { \str_if_eq_p:ee T \l_@@_vpos_block_str }
8500   { \str_if_eq_p:ee B \l_@@_vpos_block_str }
8501 }
8502 {

```

If we are in the “first column”, we must put the block as if it was with the key `r`.

```

8503 \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`.

```

8504 \bool_if:nT \g_@@_last_col_found_bool
8505 {
8506   \int_compare:nNnT { #2 } = \g_@@_col_total_int
8507     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8508 }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8509 \tl_set:Ne \l_tmpa_tl

```

```

8510     {
8511         \str_case:on \l_@@_vpos_block_str
8512     {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8513         { } {
8514             \str_case:on \l_@@_hpos_block_str
8515             {
8516                 c { center }
8517                 l { west }
8518                 r { east }
8519                 j { center }
8520             }
8521         }
8522     c {
8523         \str_case:on \l_@@_hpos_block_str
8524         {
8525             c { center }
8526             l { west }
8527             r { east }
8528             j { center }
8529         }
8530     }
8531     T {
8532         \str_case:on \l_@@_hpos_block_str
8533         {
8534             c { north }
8535             l { north-west }
8536             r { north-east }
8537             j { north }
8538         }
8539     }
8540     B {
8541         \str_case:on \l_@@_hpos_block_str
8542         {
8543             c { south }
8544             l { south-west }
8545             r { south-east }
8546             j { south }
8547         }
8548     }
8549     }
8550     }
8551     }
8552     }
8553     }
8554     \pgftransformshift
8555     {
8556         \pgfpointanchor
8557         {
8558             \@@_env: - #1 - #2 - block
8559             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8560         }
8561         \l_tmpa_tl
8562     }
8563     \pgfset { inner~sep = \c_zero_dim }
8564     \pgfnode
8565     { rectangle }
8566     \l_tmpa_tl
8567     { \box_use_drop:N \l_@@_cell_box } { } { }
8568 }

```

End of the case when `\l_@@_vpos_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```

8569     {
8570         \pgfextracty \l_tmpa_dim
8571         {
8572             \l_@@_qpoint:n
8573             {
8574                 row - \str_if_eq:eeTF b \l_@@_vpos_block_str { #3 } { #1 }
8575                 - base
8576             }
8577         }
8578         \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

8579     \pgfpointanchor
8580     {
8581         \l_@@_env: - #1 - #2 - block
8582         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8583     }
8584     {
8585         \str_case:on \l_@@_hpos_block_str
8586         {
8587             c { center }
8588             l { west }
8589             r { east }
8590             j { center }
8591         }
8592     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8593     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8594     \pgfset { inner~sep = \c_zero_dim }
8595     \pgfnode
8596     { rectangle }
8597     {
8598         \str_case:on \l_@@_hpos_block_str
8599         {
8600             c { base }
8601             l { base-west }
8602             r { base-east }
8603             j { base }
8604         }
8605     }
8606     { \box_use_drop:N \l_@@_cell_box } { } { }
8607 }
8608 \endpgfpicture
8609 }
8610 \group_end:
8611 }
8612 \cs_generate_variant:Nn \l_@@_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).

```

8613 \cs_new_protected:Npn \l_@@_add_opacity_to_fill:
8614 {
8615     \tl_if_empty:NF \l_@@_opacity_tl
8616     {
8617         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8618             {
8619                 \tl_set:Nx \l_@@_fill_tl
8620                 {
8621                     [ opacity = \l_@@_opacity_tl ,
8622                     \tl_tail:o \l_@@_fill_tl

```

```

8623     }
8624   }
8625   {
8626     \tl_set:Nc \l_@@_fill_tl
8627       { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8628   }
8629 }
8630 }

```

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*.

```

8631 \cs_new_protected:Npn \@@_stroke_block:nnnnn #1 #2 #3 #4 #5
8632 {
8633   \group_begin:
8634   \tl_clear:N \l_@@_draw_tl
8635   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8636   \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8637   \tl_if_empty:NF \l_@@_draw_tl
8638   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8639     \tl_if_eq:NnTF \l_@@_draw_tl { default }
8640       \CT@arc@
8641       { \@@_color:o \l_@@_draw_tl }
8642   }

```

The following code can't be put just before the `\pgfusepath { stroke }` (it's too late).

```

8643   \pgfsetcornersarced
8644   { \pgfpoint \l_@@_rounded_corners_dim \l_@@_rounded_corners_dim }
8645   \int_compare:nNnF { #2 } > \c@iRow
8646   {
8647     \int_compare:nNnF { #3 } > \c@jCol
8648     {
8649       \@@_qpoint:n { row - #2 }
8650       \dim_set_eq:NN \l_tmpb_dim \pgf@y
8651       \@@_qpoint:n { col - #3 }
8652       \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8653       \@@_qpoint:n
8654         { row - \int_eval:n { 1 + \int_min:nn \c@iRow { #4 } } }
8655       \dim_set_eq:NN \l_tmpa_dim \pgf@y
8656       \@@_qpoint:n
8657         { col - \int_eval:n { 1 + \int_min:nn \c@jCol { #5 } } }
8658       \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8659       \pgfpathrectanglecorners
8660         { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8661         { \pgfpoint \pgf@x \l_tmpa_dim }
8662       \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
8663         \pgfusepathqstroke
8664         { \pgfusepath { stroke } }
8665     }
8666   }
8667   \group_end:
8668 }

```

Here is the set of keys for the command `\@@_stroke_block:nnnnn`.

```

8669 \keys_define:nn { nicematrix / BlockStroke }
8670 {
8671   color .tl_set:N = \l_@@_draw_tl ,
8672   draw .code:n =
8673     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8674   draw .default:n = default ,
8675   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8676   rounded-corners .default:n = 4 pt ,
8677   rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The key `line-width` is now deprecated.

```
8678     line-width .dim_set:N = \l_@@_line_width_dim
8679 }
```

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`.

```
8680 \cs_new_protected:Npn \@@_vlines_block:nnnnnn #1 #2 #3 #4 #5 #6
8681 {
8682   \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).

```
8683   \dim_set:Nn \arrayrulewidth { #1 }
8684   \pgfsetlinewidth \arrayrulewidth % added 2026-03-28
8685   \@@_set_CTarc:n { #2 }
8686   \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).

```
8687   \seq_set_filter:Nn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
8688   {
8689     \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #3 }
8690     ||
8691     \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #4 }
8692     ||
8693     \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #5 }
8694     ||
8695     \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #6 }
8696   }

8697   \bool_if:NT \l_@@_fix_vertex_bool
8698   {
8699     \int_compare:nNnT { #4 } > 1
8700     {
8701       \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8702       {
8703         { 1 }
8704         { 1 }
8705         { \int_use:N \c@iRow }
8706         { \int_eval:n { #4 -1 } }
8707         { }
8708       }
8709     }
8710     \int_compare:nNnT { #6 } < \c@jCol
8711     {
8712       \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8713       {
8714         { 1 }
8715         { \int_eval:n { #6 + 1 } }
8716         { \int_use:N \c@iRow }
8717         { \int_use:N \c@jCol }
8718         { }
8719       }
8720     }
8721   }

8722   \int_set:Nn \l_@@_start_int { #3 }
8723   \int_set:Nn \l_@@_end_int { #5 }
8724   \int_step_inline:nnn { #4 } { #6 + 1 }
8725   {
```

```

8726     \int_set:Nn \l_@@_position_int { ##1 }
8727     \socket_use:n { nicematrix / draw-vrule }
8728   }
8729   \group_end:
8730 }

```

The command `\@@_hlines_block:nnnnnn` is used only once: in `\@@_Block_v:nnnnnn` which actually draws the block.

```

8731 \cs_new_protected:Npn \@@_hlines_block:nnnnnn #1 #2 #3 #4 #5 #6
8732 {
8733   \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).

```

8734     \dim_set:Nn \arrayrulewidth { #1 }
8735     \pgfsetlinewidth \arrayrulewidth % added 2026/03/28
8736     \@@_set_CTarc:n { #2 }
8737     \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).

```

8738     \seq_set_filter:Nn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
8739     {
8740       \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #3 }
8741       ||
8742       \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #4 }
8743       ||
8744       \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #5 }
8745       ||
8746       \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #6 }
8747     }
8748     \bool_if:NT \l_@@_fix_vertex_bool
8749     {
8750       \int_compare:nNnT { #3 } > 1
8751       {
8752         \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8753         {
8754           { 1 }
8755           { 1 }
8756           { \int_eval:n { #3 - 1 } }
8757           { \int_use:N \c@jCol }
8758           { }
8759         }
8760       }
8761       \int_compare:nNnT { #5 } < \c@iRow
8762       {
8763         \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8764         {
8765           { \int_eval:n { #5 + 1 } }
8766           { 1 }
8767           { \int_use:N \c@iRow }
8768           { \int_use:N \c@jCol }
8769           { }
8770         }
8771       }
8772     }
8773     \int_set:Nn \l_@@_start_int { #4 }
8774     \int_set:Nn \l_@@_end_int { #6 }
8775     \int_step_inline:nnn { #3 } { #5 + 1 }
8776     {

```

```

8777         \int_set:Nn \l_@@_position_int { ##1 }
8778         \socket_use:n { nicematrix / draw-hrule }
8779     }
8780 \group_end:
8781 }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke.

```

8782 \cs_new_protected:Npn \@@_stroke_borders_block:nnnn #1 #2 #3 #4 #5
8783 {
8784     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8785     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8786     \dim_compare:NnTF \l_@@_rounded_corners_dim > \c_zero_dim
8787     { \@@_error:n { borders~forbidden } }
8788     {
8789         \tl_clear_new:N \l_@@_borders_tikz_tl
8790         \keys_set:no { nicematrix / OnlyForTikzInBorders } \l_@@_borders_clist
8791         \tl_set:Nn \l_@@_tmpc_tl { #2 }
8792         \tl_set:Nn \l_@@_tmpd_tl { #3 }
8793         \tl_set:Nn \l_@@_tmpa_tl { \int_eval:n { #4 + 1 } }
8794         \tl_set:Nn \l_@@_tmpb_tl { \int_eval:n { #5 + 1 } }
8795         \@@_stroke_borders_block_i:
8796     }
8797 }

8798 \AtBeginDocument
8799 {
8800     \cs_new_protected:Npe \@@_stroke_borders_block_i:
8801     {
8802         \c_@@_pgfortikzpicture_tl
8803         \@@_stroke_borders_block_ii:
8804         \c_@@_endpgfortikzpicture_tl
8805     }
8806 }

8807 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8808 {
8809     \pgfrememberpicturepositiononpagetrue
8810     \pgf@relevantforpicturesizefalse
8811     \CT@arc@
8812     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8813     \clist_if_in:NnTF \l_@@_borders_clist { all }
8814     {
8815         \@@_stroke_vertical:n \l_@@_tmpb_tl
8816         \@@_stroke_vertical:n \l_@@_tmpd_tl
8817         \@@_stroke_horizontal:n \l_@@_tmpa_tl
8818         \@@_stroke_horizontal:n \l_@@_tmpc_tl
8819     }
8820     {
8821         \clist_if_in:NnTF \l_@@_borders_clist { right }
8822         { \@@_stroke_vertical:n \l_@@_tmpb_tl }
8823         \clist_if_in:NnTF \l_@@_borders_clist { left }
8824         { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8825         \clist_if_in:NnTF \l_@@_borders_clist { bottom }
8826         { \@@_stroke_horizontal:n \l_@@_tmpa_tl }
8827         \clist_if_in:NnTF \l_@@_borders_clist { top }
8828         { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8829     }
8830 }

8831 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8832 {
8833     tikz .code:n =
8834     \cs_if_exist:NTF \tikzpicture
8835     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }

```

```

8836     { \@@_error:n { tikz~in~borders~without~tikz } } ,
8837     tikz .value_required:n = true ,
8838     top .code:n = ,
8839     bottom .code:n = ,
8840     left .code:n = ,
8841     right .code:n = ,
8842     all .code:n = ,
8843     unknown .code:n = \@@_error:n { bad~border }
8844 }

```

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).

```

8845 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8846 {
8847   \@@_qpoint:n \l_@@_tmpc_tl
8848   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8849   \@@_qpoint:n \l_tmpa_tl
8850   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8851   \@@_qpoint:n { #1 }
8852   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8853   {
8854     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8855     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8856     \pgfusepathqstroke
8857   }
8858   {
8859     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8860     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8861   }
8862 }

```

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).

```

8863 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8864 {
8865   \@@_qpoint:n \l_@@_tmpd_tl
8866   \clist_if_in:NnTF \l_@@_borders_clist { left }
8867   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8868   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8869   \@@_qpoint:n \l_tmpb_tl
8870   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8871   \@@_qpoint:n { #1 }
8872   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8873   {
8874     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8875     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8876     \pgfusepathqstroke
8877   }
8878   {
8879     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8880     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8881   }
8882 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

8883 \keys_define:nn { nicematrix / BlockBorders }
8884 {
8885   borders .clist_set:N = \l_@@_borders_clist ,
8886   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8887   rounded-corners .default:n = 4 pt ,
8888   rules/width .dim_set:N = \l_@@_line_width_dim ,

```


The following key is deprecated.

```
8889     line-width .dim_set:N = \l_@@_line_width_dim ,
8890 }
```

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.

```
8891 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8892 {
8893   \begin { tikzpicture }
8894   \@@_clip_with_rounded_corners:
```

We use `clist_map_inline:nn` because `#5` is a list of lists.

```
8895     \clist_map_inline:nn { #1 }
8896     {
```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```
8897     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8898     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8899     (
8900     [
8901         xshift = \dim_use:N \l_@@_offset_dim ,
8902         yshift = - \dim_use:N \l_@@_offset_dim
8903     ]
8904     #2 -| #3
8905     )
8906     rectangle
8907     (
8908     [
8909         xshift = - \dim_use:N \l_@@_offset_dim ,
8910         yshift = \dim_use:N \l_@@_offset_dim
8911     ]
8912     \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8913     ) ;
8914     }
8915     \end { tikzpicture }
8916   }
8917 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

8918 \keys_define:nn { nicematrix / SpecialOffset }
8919 {
8920     offset .dim_set:N = \l_@@_offset_dim ,
8921 }
```

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.

```
8922 \cs_new_protected:Npn \@@_NullBlock:
8923 { \@@_collect_options:n { \@@_NullBlock_i: } }
8924 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8925 { }
```

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.

```
8926 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
8927 {
8928     \tl_gput_right:Ne \g_@@_pre_code_before_tl
8929     {
```

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).

```

8930 \@@_subcellcolor:nnnnnnn
8931 {
8932   \tl_if_blank:nTF { #1 }
8933   { { \exp_not:n { #2 } } }
8934   { [ #1 ] { \exp_not:n { #2 } } }
8935 }
8936 { \int_use:N \l_@@_first_row_int } % first row of the block
8937 { \int_use:N \l_@@_first_col_int } % first column of the block
8938 { \int_use:N \l_@@_last_row_int } % last row of the block
8939 { \int_use:N \l_@@_last_col_int } % last column of the block
8940 { \int_use:N \l_@@_split_int }
8941 { \int_use:N \l_@@_split_i_int }
8942 }
8943 \ignorespaces
8944 }
8945 \cs_new_protected:Npn \@@_subcellcolor:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8946 {
8947   \@@_color_opacity: #1
8948   \pgfpicture
8949   \pgf@relevantforpicturesizefalse
8950   \@@_qpoint:n { col - #3 }
8951   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8952   \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
8953   \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
8954   \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
8955   \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
8956   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8957   \@@_qpoint:n { row - #2 }
8958   \pgfpathrectanglecorners
8959   { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } \l_@@_tmpc_dim }
8960   { \pgfpoint \l_tmpb_dim \pgf@y }
8961   \pgfusepathqfill
8962   \endpgfpicture
8963 }

```

27 Automatic arrays

We will extract some keys and pass the other keys to the environment {NiceArrayWithDelims}.

```

8964 \keys_define:nn { nicematrix / Auto }
8965 {
8966   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8967   columns-type .value_required:n = true ,
8968   l .meta:n = { columns-type = l } ,
8969   r .meta:n = { columns-type = r } ,
8970   c .meta:n = { columns-type = c } ,
8971   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8972   delimiters / color .value_required:n = true ,
8973   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8974   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8975   delimiters .value_required:n = true ,
8976   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8977   rounded-corners .default:n = 4 pt
8978 }
8979 \NewDocumentCommand \AutoNiceMatrixWithDelims
8980 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8981 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

```

```

8982 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8983 {

```

The group is for the protection of the keys.

```

8984 \group_begin:
8985 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8986 \use:e
8987 {
8988   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8989   { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8990   [ \exp_not:o \l_tmpa_tl ]
8991 }
8992 \int_if_zero:nT \l_@@_first_row_int
8993 {
8994   \int_if_zero:nT \l_@@_first_col_int { & }
8995   \prg_replicate:nn { #4 - 1 } { & }
8996   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8997 }
8998 \prg_replicate:nn { #3 }
8999 {
9000   \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).

```

9001   \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
9002   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9003 }
9004 \int_compare:nNnT \l_@@_last_row_int > { -2 }
9005 {
9006   \int_if_zero:nT \l_@@_first_col_int { & }
9007   \prg_replicate:nn { #4 - 1 } { & }
9008   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9009 }
9010 \end { NiceArrayWithDelims }
9011 \group_end:
9012 }
9013 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
9014 {
9015   \cs_set_protected:cpn { #1 AutoNiceMatrix }
9016   {
9017     \bool_gset_true:N \g_@@_delims_bool
9018     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
9019     \AutoNiceMatrixWithDelims { #2 } { #3 }
9020   }
9021 }

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

9022 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
9023 {
9024   \group_begin:
9025   \bool_gset_false:N \g_@@_delims_bool
9026   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
9027   \group_end:
9028 }

```

28 The redefinition of the command \dotfill

```

9029 \cs_set_eq:NN \@@_old_dotfill: \dotfill
9030 \cs_new_protected:Npn \@@_dotfill:
9031 {

```

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}`).

```

9032   \@@_old_dotfill:
9033   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
9034 }

```

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`.

```

9035 \cs_new_protected:Npn \@@_dotfill_i:
9036 {
9037   \dim_compare:nNtT { \box_wd:N \l_@@_cell_box } = \c_zero_dim
9038   { \@@_old_dotfill: }
9039 }

```

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.

```

9040 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
9041 {
9042   \tl_gput_right:Ne \g_@@_rules_tl
9043   {
9044     \@@_draw_diagbox:nnnnnn
9045     { \int_use:N \c@iRow }
9046     { \int_use:N \c@jCol }
9047     { \int_use:N \c@iRow }
9048     { \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.

```

9049     { \g_@@_row_style_tl \exp_not:n { #1 } }
9050     { \g_@@_row_style_tl \exp_not:n { #2 } }
9051   }

```

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.

```

9052   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
9053   {
9054     { \int_use:N \c@iRow }
9055     { \int_use:N \c@jCol }
9056     { \int_use:N \c@iRow }
9057     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

9058     { }
9059   }
9060 }

```

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.

```

9061 \cs_new_protected:Npn \@@_draw_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
9062 {

```

We *must* use an environment `{pgfscope}` and not a simple group of TeX.

```

9063 \begin { pgfscope }
9064 \@@_qpoint:n { row - #1 }
9065 \dim_set_eq:NN \l_tmpa_dim \pgf@y
9066 \@@_qpoint:n { col - #2 }
9067 \dim_set_eq:NN \l_tmpb_dim \pgf@x
9068 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
9069 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
9070 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
9071 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
9072 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
9073 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
9074 {

```

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.

```

9075 \CT@arc@
9076 \pgfsetroundcap
9077 \pgfusepathqstroke
9078 }
9079 \pgfset { inner~sep = 1 pt }
9080 \pgfscope
9081 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
9082 \pgfnode { rectangle } { south~west }
9083 {
9084 \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

9085 \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
9086 \end { minipage }
9087 }
9088 { }
9089 { }
9090 \endpgfscope
9091 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
9092 \pgfnode { rectangle } { north~east }
9093 {
9094 \begin { minipage } { 20 cm }
9095 \raggedleft
9096 \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
9097 \end { minipage }
9098 }
9099 { }
9100 { }
9101 \end { pgfscope }
9102 }

```

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. 89.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

9103 \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 `\`.

```

9104 \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`.

```

9105 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
9106 {
9107   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
9108   \@@_CodeAfter_iv:n
9109 }

```

We catch the argument of the command `\end` (in `#1`).

```

9110 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
9111 {

```

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.

```

9112   \str_if_eq:eeTF \currenvir { #1 }
9113   { \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`.

```

9114   {
9115     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
9116     \@@_CodeAfter_ii:n
9117   }
9118 }

```

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.

```

9119 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
9120 {
9121   \pgfpicture
9122   \pgfrememberpicturepositiononpagetrue
9123   \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.

```

9124   \@@_qpoint:n { row - 1 }
9125   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
9126   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
9127   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y

```

We will compute in `\l_tmpa_dim` the x -value where we will have to put our delimiter (on the left side or on the right side).

```

9128   \bool_if:nTF { #3 }
9129   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
9130   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
9131   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
9132   {
9133     \cs_if_exist:cT
9134     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }

```

```

9135     {
9136         \pgfpointanchor
9137         { \l_@@_env: - ##1 - #2 }
9138         { \bool_if:nTF { #3 } { west } { east } }
9139         \dim_set:Nn \l_tmpa_dim
9140         {
9141             \bool_if:nTF { #3 }
9142             \dim_min:nn
9143             \dim_max:nn
9144             \l_tmpa_dim
9145             \pgf@x
9146         }
9147     }
9148 }

```

Now we can put the delimiter with a node of PGF.

```

9149 \pgfset { inner~sep = \c_zero_dim }
9150 \dim_zero:N \nulldelimiterspace
9151 \pgftransformshift
9152 {
9153     \pgfpoint
9154     \l_tmpa_dim
9155     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
9156 }
9157 \pgfnode
9158 { rectangle }
9159 { \bool_if:nTF { #3 } { east } { west } }
9160 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

9161     \nullfont
9162     $ % $
9163     \l_@@_color:o \l_@@_delimiters_color_tl
9164     \bool_if:nTF { #3 } { \left #1 } { \left . }
9165     \vcenter
9166     {
9167         \nullfont
9168         \hrule \@height
9169             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
9170             \@depth \c_zero_dim
9171             \@width \c_zero_dim
9172     }
9173     \bool_if:nTF { #3 } { \right . } { \right #1 }
9174     $ % $
9175 }
9176 { }
9177 { }
9178 \endpgfpicture
9179 }

```

32 The command `\SubMatrix`

```

9180 \keys_define:nn { nicematrix / sub-matrix }
9181 {
9182     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
9183     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
9184     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
9185     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
9186     xshift .value_required:n = true ,
9187     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9188     delimiters / color .value_required:n = true ,

```

```

9189     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
9190     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9191     hlines .default:n = all ,
9192     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9193     vlines .default:n = all ,
9194     hvlines .meta:n = { hlines, vlines } ,
9195     hvlines .value_forbidden:n = true
9196   }
9197 \keys_define:nn { nicematrix }
9198 {
9199     SubMatrix .inherit:n = nicematrix / sub-matrix ,
9200     NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9201     pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9202     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9203 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

9204 \keys_define:nn { nicematrix / SubMatrix }
9205 {
9206     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9207     delimiters / color .value_required:n = true ,
9208     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9209     hlines .default:n = all ,
9210     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9211     vlines .default:n = all ,
9212     hvlines .meta:n = { hlines, vlines } ,
9213     hvlines .value_forbidden:n = true ,
9214     name .code:n =
9215         \tl_if_empty:nTF { #1 }
9216         { \@@_error:n { Invalid-name } }
9217         {
9218             \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
9219             {
9220                 \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
9221                 { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
9222                 {
9223                     \str_set:Nn \l_@@_submatrix_name_str { #1 }
9224                     \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
9225                 }
9226             }
9227             { \@@_error:n { Invalid-name } }
9228         } ,
9229     name .value_required:n = true ,
9230     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
9231     rules .value_required:n = true ,
9232     code .tl_set:N = \l_@@_code_tl ,
9233     code .value_required:n = true ,
9234     unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
9235 }

9236 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
9237 {
9238     \tl_gput_right:Ne \g_@@_pre_code_after_tl
9239     {
9240         \SubMatrix { #1 } { #2 } { #3 } { #4 }
9241         [
9242             delimiters / color = \l_@@_delimiters_color_tl ,
9243             hlines = \l_@@_submatrix_hlines_clist ,
9244             vlines = \l_@@_submatrix_vlines_clist ,
9245             extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
9246             left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
9247             right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,

```



```

9248         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
9249         #5
9250     ]
9251 }
9252 \@@_SubMatrix_in_code_before_i { #2 } { #3 }
9253 \ignorespaces
9254 }
9255 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
9256 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9257 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
9258 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
9259 {
9260     \seq_gput_right:Ne \g_@@_submatrix_seq
9261     {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

9262     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
9263     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
9264     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
9265     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
9266 }
9267 }

```

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).

```

9268 \NewDocumentCommand \@@_compute_i_j:nn
9269 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9270 { \@@_compute_i_j:nnnn #1 #2 }
9271 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
9272 {
9273     \def \l_@@_first_i_tl { #1 }
9274     \def \l_@@_first_j_tl { #2 }
9275     \def \l_@@_last_i_tl { #3 }
9276     \def \l_@@_last_j_tl { #4 }
9277     \tl_if_eq:NnT \l_@@_first_i_tl { last }
9278     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
9279     \tl_if_eq:NnT \l_@@_first_j_tl { last }
9280     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
9281     \tl_if_eq:NnT \l_@@_last_i_tl { last }
9282     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
9283     \tl_if_eq:NnT \l_@@_last_j_tl { last }
9284     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
9285 }

```

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`.

```

9286 \AtBeginDocument
9287 {
9288   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m O { } E { _ ^ } { { } { } } }
9289   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
9290     { \@@_sub_matrix:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
9291 }
9292 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
9293 {
9294   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

9295 \@@_compute_i_j:nn { #2 } { #3 }
9296 \int_compare:nNt \l_@@_first_i_tl = \l_@@_last_i_tl
9297 { \def \arraystretch { 1 } }
9298 \bool_lazy_or:nnTF
9299 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9300 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9301 { \@@_error:nn { Construct-too-large } { \SubMatrix } }
9302 {
9303   \str_clear_new:N \l_@@_submatrix_name_str
9304   \keys_set:nn { nicematrix / SubMatrix } { #5 }
9305   \pgfpicture
9306   \pgfrememberpicturepositiononpagetrue
9307   \pgf@relevantforpicturesizefalse
9308   \pgfset { inner~sep = \c_zero_dim }
9309   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9310   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curryfication.

```

9311 \bool_if:NtF \l_@@_submatrix_slim_bool
9312 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
9313 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
9314 {
9315   \cs_if_exist:cT
9316   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9317   {
9318     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9319     \dim_compare:nNt \pgf@x < \l_@@_x_initial_dim
9320     { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9321   }
9322   \cs_if_exist:cT
9323   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9324   {
9325     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9326     \dim_compare:nNt \pgf@x > \l_@@_x_final_dim
9327     { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9328   }
9329 }
9330 \dim_compare:nNtF \l_@@_x_initial_dim = \c_max_dim
9331 { \@@_impossible_delimiter:n { left } }
9332 {
9333   \dim_compare:nNtF \l_@@_x_final_dim = { - \c_max_dim }
9334   { \@@_impossible_delimiter:n { right } }
9335   { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
9336 }
9337 \endpgfpicture
9338 }
9339 \group_end:
9340 \ignorespaces
9341 }

```

The argument of the following command will be provided by curryfication.

```

9342 \cs_new_protected:Npn \@@_impossible_delimiter:n #1
9343 {
9344   \bool_if:NTF \l_@@_no_cell_nodes_bool
9345     { \@@_error:n { Impossible~SubMatrix~no~cell~nodes } }
9346     { \@@_error:nn { Impossible~SubMatrix } { #1 } }
9347 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

9348 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
9349 {
9350   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
9351   \dim_set:Nn \l_@@_y_initial_dim
9352   {
9353     \fp_to_dim:n
9354     {
9355       \pgf@y
9356       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
9357     }
9358   }
9359   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
9360   \dim_set:Nn \l_@@_y_final_dim
9361   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
9362   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
9363   {
9364     \cs_if_exist:cT
9365     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
9366     {
9367       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
9368       \dim_set:Nn \l_@@_y_initial_dim
9369       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
9370     }
9371     \cs_if_exist:cT
9372     { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
9373     {
9374       \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
9375       \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
9376       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
9377     }
9378   }
9379   \dim_set:Nn \l_tmpa_dim
9380   {
9381     \l_@@_y_initial_dim - \l_@@_y_final_dim +
9382     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
9383   }
9384   \dim_zero:N \nullldelimiterspace

```

We will draw the rules in the \SubMatrix.

```

9385 \group_begin:
9386 \pgfsetlinewidth { 1.1 \arrayrulewidth }
9387 \@@_set_CTarc:o \l_@@_rules_color_tl
9388 \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`.

```

9389 \seq_map_inline:Nn \g_@@_cols_vlism_seq
9390 {
9391   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
9392   {
9393     \int_compare:nNnT
9394     { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
9395     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

9396         \@@_qpoint:n { col - ##1 }
9397         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9398         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9399         \pgfusepathqstroke
9400     }
9401 }
9402 }

```

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.

```

9403     \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
9404     { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
9405     { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
9406     {
9407         \bool_lazy_and:nnTF
9408         { \int_compare_p:nNn { ##1 } > \c_zero_int }
9409         {
9410             \int_compare_p:nNn
9411             { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
9412         {
9413             \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
9414             \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9415             \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9416             \pgfusepathqstroke
9417         }
9418         { \@@_error:mn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
9419     }

```

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.

```

9420     \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
9421     { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
9422     { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
9423     {
9424         \bool_lazy_and:nnTF
9425         { \int_compare_p:nNn { ##1 } > \c_zero_int }
9426         {
9427             \int_compare_p:nNn
9428             { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
9429         {
9430             \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

9431     \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

9432     \dim_set:Nn \l_tmpa_dim
9433     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9434     \str_case:nn { #1 }
9435     {
9436         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9437         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
9438         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9439     }
9440     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

9441     \dim_set:Nn \l_tmpb_dim
9442     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9443     \str_case:nn { #2 }
9444     {
9445         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9446         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }

```

```

9447         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9448     }
9449     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9450     \pgfusepath{stroke}
9451     \group_end:
9452 }
9453 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
9454 }

```

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).

```

9455 \str_if_empty:NF \l_@@_submatrix_name_str
9456 {
9457     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
9458     \l_@@_x_initial_dim \l_@@_y_initial_dim
9459     \l_@@_x_final_dim \l_@@_y_final_dim
9460 }
9461 \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`.

```

9462 \begin { pgfscope }
9463 \pgftransformshift
9464 {
9465     \pgfpoint
9466     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9467     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9468 }
9469 \str_if_empty:NTF \l_@@_submatrix_name_str
9470 { \@@_node_left:nn #1 { } }
9471 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9472 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9473 \pgftransformshift
9474 {
9475     \pgfpoint
9476     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9477     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9478 }
9479 \str_if_empty:NTF \l_@@_submatrix_name_str
9480 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9481 {
9482     \@@_node_right:nnnn #2
9483     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9484 }

```

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`.

```

9485 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9486 \flag_clear_new:N \l_@@_code_flag
9487 \l_@@_code_tl
9488 }

```

In the key code of the command `\SubMatrix` there may be TikZ instructions. We want that, in these instructions, the i and j in specifications of nodes of the forms $i-j$, $\text{row-}i$, $\text{col-}j$ and $i-|j$ refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

9489 \cs_set_eq:NN \@@_old_pgfpntanchor: \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of TikZ nodes which are computed in an expandable way.

The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by curryfication.

```

9490 \cs_new:Npn \@@_pgfpointanchor:n #1
9491 { \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`).

```

9492 \cs_new:Npn \@@_pgfpointanchor_i:n #1
9493 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }

9494 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9495 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9496 \str_if_empty:nTF { #1 }

```

First, when the name of the name begins with `\tikz@pp@name`.

```

9497 { \@@_pgfpointanchor_iv:w #2 }

```

And now, when there is no `\tikz@pp@name`.

```

9498 { \@@_pgfpointanchor_ii:n { #1 } }
9499 }

```

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`).

```

9500 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
9501 { \@@_pgfpointanchor_ii:n { #1 } }

```

With the command `\@@_pgfpointanchor_ii:n`, we deal with the actual name of the node (without the `\tikz@pp@name`). First, we have to detect whether it is of the form `i` of the form `i-j` (with an hyphen).

Remark: It would be possible to test the presence of the hyphen in an expandable way to using `\etl_if_in:nnTF` of the package `etl` but, as of now, we do not load `etl`.

```

9502 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

9503 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
9504 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9505 \str_if_empty:nTF { #2 }

```

First the case where the argument does *not* contain an hyphen.

```

9506 { \@@_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`).

```

9507 { \@@_pgfpointanchor_iii:w { #1 } #2 }
9508 }

```

The following function is for the case when the name contains an hyphen.

```

9509 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9510 {

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9511 \@@_env:
9512 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9513 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9514 }

```

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.

```

9515 \tl_const:Nn \c_@@_integers_alist_tl
9516 {
9517   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9518   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9519   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9520   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9521 }

9522 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9523 {

```

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`.

```

9524 \str_case:nVTF { #1 } \c_@@_integers_alist_tl
9525 {
9526   \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9527 \@@_env: -
9528 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9529 { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9530 { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9531 }
9532 {
9533   \str_if_eq:eeTF { #1 } { last }
9534   {
9535     \flag_raise:N \l_@@_code_flag
9536     \@@_env: -
9537     \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9538     { \int_eval:n { \l_@@_last_i_tl + 1 } }
9539     { \int_eval:n { \l_@@_last_j_tl + 1 } }
9540   }
9541   { #1 }
9542 }
9543 }

```

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`).

```

9544 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9545 {
9546   \pgfnode
9547   { rectangle }
9548   { east }
9549   {
9550     \nullfont
9551     $ % $
9552     \@@_color:o \l_@@_delimiters_color_tl
9553     \left #1
9554     \vcenter
9555     {
9556       \nullfont
9557       \hrule \@height \l_tmpa_dim
9558       \@depth \c_zero_dim

```

```

9559             \@width \c_zero_dim
9560         }
9561         \right .
9562         $ % $
9563     }
9564     { #2 }
9565     { }
9566 }

```

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.

```

9567 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9568 {
9569     \pgfnode
9570     { rectangle }
9571     { west }
9572     {
9573         \nullfont
9574         $ % $
9575         \colorlet { current-color } { . }
9576         \@@_color:o \l_@@_delimiters_color_tl
9577         \left .
9578         \vcenter
9579         {
9580             \nullfont
9581             \hrule \@height \l_tmpa_dim
9582                 \@depth \c_zero_dim
9583                 \@width \c_zero_dim
9584         }
9585         \right #1
9586         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9587         ^ { \color { current-color } \smash { #4 } }
9588         $ % $
9589     }
9590     { #2 }
9591     { }
9592 }

```

33 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9593 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9594 {
9595     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9596     \ignorespaces
9597 }
9598 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9599 {
9600     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9601     \ignorespaces
9602 }
9603 \keys_define:nn { nicematrix / Brace }
9604 {
9605     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9606     left-shorten .value_forbidden:n = true ,
9607     right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,

```



```

9608 right-shorten .value_forbidden:n = true ,
9609 shorten .meta:n = { left-shorten , right-shorten } ,
9610 shorten .value_forbidden:n = true ,
9611 yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9612 color .tl_set:N = \l_tmpa_tl ,
9613 color .value_required:n = true ,
9614 unknown .code:n =
9615     \@@_unknown_key:nn
9616     { nicematrix / Brace }
9617     { Unknown-key-for-Brace }
9618 }

```

#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`.

```

9619 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9620 {
9621     \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9622     \@@_compute_i_j:nn { #1 } { #2 }
9623     \bool_lazy_or:nnTF
9624     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9625     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9626     {
9627         \str_if_eq:eeTF { #5 } { under }
9628         { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9629         { \@@_error:nn { Construct-too-large } { \OverBrace } }
9630     }
9631     {
9632         \tl_clear:N \l_tmpa_tl
9633         \keys_set:nn { nicematrix / Brace } { #4 }
9634         \tl_if_empty:NF \l_tmpa_tl { \color \l_tmpa_tl }
9635         \pgfpicture
9636         \pgfrememberpicturerepositiononpagetrue
9637         \pgf@relevantforpicturesizefalse
9638         \bool_if:NT \l_@@_brace_left_shorten_bool
9639         {
9640             \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9641             \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9642             {
9643                 \cs_if_exist:cT
9644                 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9645                 {
9646                     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9647
9648                     \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9649                     { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9650                 }
9651             }
9652         }
9653         \bool_lazy_or:nnT
9654         { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9655         { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
9656         {
9657             \@@_qpoint:n { col - \l_@@_first_j_tl }
9658             \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9659         }
9660         \bool_if:NT \l_@@_brace_right_shorten_bool
9661         {
9662             \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9663             \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9664             {
9665                 \cs_if_exist:cT

```

```

9666         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9667         {
9668             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9669             \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9670             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9671         }
9672     }
9673 }
9674 \bool_lazy_or:nnT
9675 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9676 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
9677 {
9678     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9679     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9680 }
9681 \pgfset { inner~sep = \c_zero_dim }
9682 \str_if_eq:eeTF { #5 } { under }
9683 { \@@_underbrace_i:n { #3 } }
9684 { \@@_overbrace_i:n { #3 } }
9685 \endpgfpicture
9686 }
9687 \group_end:
9688 }

```

The argument is the text to put above the brace.

```

9689 \cs_new_protected:Npn \@@_overbrace_i:n #1
9690 {
9691     \@@_qpoint:n { row - \l_@@_first_i_tl }
9692     \pgftransformshift
9693     {
9694         \pgfpoint
9695         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9696         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9697     }
9698     \pgfnode
9699     { rectangle }
9700     { south }
9701     {
9702         \vtop
9703         {
9704             \group_begin:
9705             \everycr { }
9706             \halign
9707             {
9708                 \hfil ## \hfil \crcr
9709                 \bool_if:NTF \l_@@_tabular_bool
9710                 { \begin { tabular } { c } #1 \end { tabular } }
9711                 { $ \begin { array } { c } #1 \end { array } $ }
9712                 \cr
9713                 $ % $
9714                 \overbrace
9715                 {
9716                     \hbox_to_wd:nn
9717                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9718                     { }
9719                 }
9720                 $ % $
9721                 \cr
9722             }
9723             \group_end:
9724         }
9725     }
9726     { }
9727     { }

```

```
9728 }
```

The argument is the text to put under the brace.

```
9729 \cs_new_protected:Npn \@@_underbrace_i:n #1
9730 {
9731   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9732   \pgftransformshift
9733   {
9734     \pgfpoint
9735     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9736     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9737   }
9738   \pgfnode
9739   { rectangle }
9740   { north }
9741   {
9742     \group_begin:
9743     \everycr { }
9744     \vbox
9745     {
9746       \halign
9747       {
9748         \hfil ## \hfil \crcr
9749         $ % $
9750         \underbrace
9751         {
9752           \hbox_to_wd:nn
9753           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9754           { }
9755         }
9756         $ % $
9757         \cr
9758         \bool_if:NTF \l_@@_tabular_bool
9759         { \begin { tabular } { c } #1 \end { tabular } }
9760         { $ \begin { array } { c } #1 \end { array } $ }
9761         \cr
9762       }
9763     }
9764     \group_end:
9765   }
9766   { }
9767   { }
9768 }
```

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`).

```
9769 \AddToHook { package / tikz / after }
9770 {
9771   \tikzset
9772   {
9773     nicematrix / brace / .style =
9774     {
```

```

9775         decoration = { brace , raise = -0.15 em } ,
9776         decorate ,
9777     } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9778     nicematrix / mirrored-brace / .style =
9779     {
9780         nicematrix / brace ,
9781         decoration = mirror ,
9782     }
9783 }
9784 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9785 \keys_define:nn { nicematrix / Hbrace }
9786 {
9787     color .code:n = ,
9788     horizontal-label .code:n = ,
9789     horizontal-labels .code:n = ,
9790     shorten .code:n = ,
9791     shorten-start .code:n = ,
9792     shorten-end .code:n = ,
9793     shorten+ .code:n = ,
9794     shorten-start+ .code:n = ,
9795     shorten-end+ .code:n = ,
9796     shorten~+ .code:n = ,
9797     shorten-start~+ .code:n = ,
9798     shorten-end~+ .code:n = ,
9799     brace-shift .code:n = ,
9800     brace-shift+ .code:n = ,
9801     brace-shift~+ .code:n = ,
9802     unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9803 }

```

Here we need an “fully expandable” command.

```

9804 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9805 {
9806     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9807     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9808     { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9809 }

```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

9810 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9811 {
9812     \int_compare:nNnTF \c@iRow < { 2 }
9813     {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9814     \str_if_eq:nnTF { #2 } { * }
9815     {
9816         \bool_set_true:N \l_@@_nullify_dots_bool
9817         \Ldots
9818         [
9819             line-style = nicematrix / brace ,
9820             #1 ,
9821             up =
9822             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9823         ]
9824     }

```

```

9825     {
9826         \Hdotsfor
9827         [
9828             line-style = nicematrix / brace ,
9829             #1 ,
9830             up =
9831             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9832         ]
9833         { #2 }
9834     }
9835 }
9836 {
9837     \str_if_eq:nnTF { #2 } { * }
9838     {
9839         \bool_set_true:N \l_@@_nullify_dots_bool
9840         \Ldots
9841         [
9842             line-style = nicematrix / mirrored-brace ,
9843             #1 ,
9844             down =
9845             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9846         ]
9847     }
9848     {
9849         \Hdotsfor
9850         [
9851             line-style = nicematrix / mirrored-brace ,
9852             #1 ,
9853             down =
9854             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9855         ]
9856         { #2 }
9857     }
9858 }
9859 \keys_set:nn { nicematrix / Hbrace } { #1 }
9860 }

```

```

9861 \NewDocumentCommand { \@@_Vbrace } { 0 { } m m }
9862 {
9863     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9864     { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9865     { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9866 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

9867 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
9868 {
9869     \int_compare:nNnTF \c@jCol < { 2 }
9870     {
9871         \str_if_eq:nnTF { #2 } { * }
9872         {
9873             \bool_set_true:N \l_@@_nullify_dots_bool
9874             \Vdots
9875             [
9876                 Vbrace ,
9877                 line-style = nicematrix / mirrored-brace ,
9878                 #1 ,
9879                 down =
9880                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9881             ]
9882         }
9883     }

```

```

9884         \Vdotsfor
9885         [
9886             Vbrace ,
9887             line-style = nicematrix / mirrored-brace ,
9888             #1 ,
9889             down =
9890                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9891         ]
9892     { #2 }
9893 }
9894 }
9895 {
9896     \str_if_eq:nnTF { #2 } { * }
9897     {
9898         \bool_set_true:N \l_@@_nullify_dots_bool
9899         \Vdots
9900         [
9901             Vbrace ,
9902             line-style = nicematrix / brace ,
9903             #1 ,
9904             up =
9905                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9906         ]
9907     }
9908     {
9909         \Vdotsfor
9910         [
9911             Vbrace ,
9912             line-style = nicematrix / brace ,
9913             #1 ,
9914             up =
9915                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9916         ]
9917         { #2 }
9918     }
9919 }
9920 \keys_set:nn { nicematrix / Hbrace } { #1 }
9921 }

```

35 The command TikzEveryCell

```

9922 \bool_new:N \l_@@_not_empty_bool
9923 \bool_new:N \l_@@_empty_bool
9924
9925 \keys_define:nn { nicematrix / TikzEveryCell }
9926 {
9927     not-empty .code:n =
9928         \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
9929         { \bool_set_true:N \l_@@_not_empty_bool }
9930         { \@@_error:n { detection-of-empty-cells } } ,
9931     not-empty .value_forbidden:n = true ,
9932     empty .code:n =
9933         \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
9934         { \bool_set_true:N \l_@@_empty_bool }
9935         { \@@_error:n { detection-of-empty-cells } } ,
9936     empty .value_forbidden:n = true ,
9937     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
9938 }
9939

```

```

9940
9941 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9942 {
9943   \IfPackageLoadedTF { tikz }
9944   {
9945     \group_begin:
9946     \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.

```

9947     \tl_set:Nn \l_tmpa_tl { { #2 } }
9948     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9949     { \@@_for_a_block:nnnnn ##1 }
9950     \@@_all_the_cells:
9951     \group_end:
9952   }
9953   { \@@_error:n { TikzEveryCell~without~tikz } }
9954 }
9955
9956
9957 \cs_new_protected:Nn \@@_all_the_cells:
9958 {
9959   \int_step_inline:nn \c@iRow
9960   {
9961     \int_step_inline:nn \c@jCol
9962     {
9963       \cs_if_exist:cF { cell - ##1 - #####1 }
9964       {
9965         \clist_if_in:Nef \l_@@_corners_cells_clist
9966         { ##1 - #####1 }
9967         {
9968           \bool_set_false:N \l_tmpa_bool
9969           \cs_if_exist:cTF
9970           { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
9971           {
9972             \bool_if:NF \l_@@_empty_bool
9973             { \bool_set_true:N \l_tmpa_bool }
9974           }
9975           {
9976             \bool_if:NF \l_@@_not_empty_bool
9977             { \bool_set_true:N \l_tmpa_bool }
9978           }
9979           \bool_if:NT \l_tmpa_bool
9980           {
9981             \@@_block_tikz:nnnnn
9982             \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
9983           }
9984         }
9985       }
9986     }
9987   }
9988 }
9989
9990 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9991 {
9992   \bool_if:NF \l_@@_empty_bool
9993   {
9994     \@@_block_tikz:nnnnn
9995     \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9996   }
9997   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9998 }
9999
10000 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn

```

```

10001 {
10002   \int_step_inline:nnn { #1 } { #3 }
10003   {
10004     \int_step_inline:nnn { #2 } { #4 }
10005     { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
10006   }
10007 }

```

36 The key draw-trees-in-col

```

10008 \cs_new_protected:Npn \@@_draw_trees:
10009 {
10010   \bool_if:NTF \l_@@_no_cell_nodes_bool
10011   { \@@_error:n { draw-trees-with-no-cell-nodes } }
10012   \@@_draw_trees_i:
10013 }
10014 \cs_new_protected:Npn \@@_draw_trees_i:
10015 {
10016   \@@_expand_clist_hvlines:NN \g_@@_col_with_trees_clist \c@jCol
10017   \dim_zero_new:N \l_@@_em_dim
10018   \dim_set:Nn \l_@@_em_dim { 1 em }
10019   \dim_zero_new:N \l_@@_ex_dim
10020   \dim_set:Nn \l_@@_ex_dim { 1 ex }
10021   \pgfpicture
10022   \pgfrememberpicturepositiononpagetrue
10023   \pgf@relevantforpicturesizefalse
10024   \dim_compare:nNnT \l_@@_trees_line_width_dim > \c_zero_dim
10025   { \pgfsetlinewidth { \l_@@_trees_line_width_dim } }
10026   \@@_color:o \l_@@_trees_color_tl
10027   \pgfsetcornersarced
10028   { \pgfpoint \l_@@_trees_rounded_corners_dim \l_@@_trees_rounded_corners_dim }
10029   \clist_map_function:NN \g_@@_col_with_trees_clist
10030   \@@_draw_trees_in_col:n
10031   \clist_gclear:N \g_@@_col_with_trees_clist
10032   \endpgfpicture
10033 }
10034 \cs_new_protected:Npn \@@_draw_trees_in_col:n #1
10035 {

```

The argument is provided by curryfication.

```

10036   \int_compare:nNnTF { #1 } > \c@jCol
10037   { \@@_error:nn { Col~outside~tabular~in~trees } }
10038   {
10039     \int_compare:nNnTF { #1 } = \c@jCol
10040     { \@@_error:nn { Last-col-in-trees } }
10041     \@@_draw_trees_in_col_i:n
10042   }
10043   { #1 }
10044 }
10045 \cs_new_protected:Npn \@@_draw_trees_in_col_i:n #1
10046 {
10047   \int_set:Nn \l_tmpa_int { 1 }
10048   \int_step_inline:nn { \c@iRow + 1 }
10049   {
10050     \cs_if_exist:cT
10051     { pgf @ sh @ ns @ \@@_env: - ##1 - #1 }
10052     {
10053       \int_compare:nNnT { ##1 } > { \l_tmpa_int + 1 }
10054       {

```


Now, you will, potentially, draw a tree.

```

10055         \@@_draw_tree:nee
10056         { #1 }
10057         { \int_use:N \l_tmpa_int }
10058         { \int_eval:n { ##1 - 1 } }
10059     }
10060     \int_set:Nn \l_tmpa_int { ##1 }
10061 }
10062 }
10063 \int_compare:nNnT \c@iRow > \l_tmpa_int
10064 {
10065     \@@_draw_tree:nee
10066     { #1 }
10067     { \int_use:N \l_tmpa_int }
10068     { \int_eval:n { \c@iRow } }
10069 }
10070 }

```

#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.

```

10071 \cs_new_protected:Npn \@@_draw_tree:nnn #1 #2 #3
10072 {

```

\l_tmpa_dim will be the x -value of the vertical rule that we will draw.

```

10073     \pgfpointanchor { \@@_env: - #1 } { 5 }
10074     \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.

```

10075     \bool_set_false:N \l_tmpa_bool
10076     \int_step_inline:nnnn { #3 } { -1 } { #2 + 1 }
10077     {
10078         \cs_if_exist:cT
10079         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #1 + 1 } }

```

We have found the last branch to draw.

```

10080     {
10081         \pgfpointanchor
10082         { \@@_env: - ##1 - \int_eval:n { #1 + 1 } }
10083         { base~west }
10084         \pgfpathmoveto
10085         {
10086             \pgfpoint
10087             { \dim_eval:n { \pgf@x - 0.7 \l_@@_ex_dim } }
10088             { \dim_eval:n { \pgf@y + 0.25 \l_@@_em_dim } }
10089         }
10090         \pgfpathlineto { \pgfpoint \l_tmpa_dim \pgf@y }
10091         \bool_if:NF \l_tmpa_bool
10092         {
10093             \pgfpointanchor{ \@@_env: - #2 - #1 } { south }
10094             \pgfpathlineto
10095             { \pgfpoint \l_tmpa_dim { \dim_eval:n { \pgf@y - 3pt } } }
10096             \bool_set_true:N \l_tmpa_bool
10097         }
10098         \pgfusepath { stroke }
10099     }
10100 }
10101 }
10102 \cs_generate_variant:Nn \@@_draw_tree:nnn { n e e }

```

37 The key create-blocks-in-col

```

10103 \cs_new_protected:Npn \@@_create_blocks_in_col:

```

```

10104 {
10105   \@@_expand_clist_hvlines:NN \g_@@_cbic_clist \c@jCol
10106   \clist_map_inline:Nn \g_@@_cbic_clist
10107   {
10108     \cs_set:cpn
10109     {
10110       pgf @ sh @ ns @ \@@_env:
10111       - \int_eval:n { \c@iRow + 1 } - ##1 }
10112     { rien }
10113   }
10114   \int_set:Nn \l_tmpa_int { 1 }
10115   \int_step_inline:nn { \c@iRow + 1 }
10116   {
10117     \cs_if_exist:cT
10118     { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 }
10119     {
10120       \int_compare:nNnT { #####1 } > { \l_tmpa_int + 1 }
10121       {
10122         \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
10123         {
10124           { \int_use:N \l_tmpa_int }
10125           { ##1 }
10126           { \int_eval:n { #####1 - 1 } }
10127           { ##1 }
10128           { }
10129         }
10130       }
10131       \int_set:Nn \l_tmpa_int { #####1 }
10132     }
10133   }
10134   \clist_gclear:N \g_@@_cbic_clist
10135 }

```

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.

```

10136 \cs_new_protected:Npn \@@_ShowCellNamesCodeBefore
10137 { \tl_put_right:Nn \@@_actually_color: \@@_ShowCellNames } % noqa
10138 \NewDocumentCommand \@@_ShowCellNames { }
10139 {
10140   \bool_if:NT \l_@@_in_code_after_bool
10141   {
10142     \pgfpicture
10143     \pgfrememberpicturerepositiononpagetrue
10144     \pgf@relevantforpicturesizefalse
10145     \pgfpathrectanglecorners
10146     { \@@_qpoint:n { 1 } }
10147     { \@@_qpoint:n { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } } }
10148     \pgfsetfillopacity { 0.75 }
10149     \pgfsetfillcolor { white }
10150     \pgfusepathqfill
10151     \endpgfpicture
10152   }
10153   \dim_gzero_new:N \g_@@_tmpc_dim

```

```

10154 \dim_gzero_new:N \g_@@_tmpd_dim
10155 \dim_gzero_new:N \g_@@_tmpe_dim
10156 \int_step_inline:nn \c@iRow
10157 {
10158   \bool_if:NTF \l_@@_in_code_after_bool
10159   {
10160     \pgfpicture
10161     \pgfrememberpicturepositiononpagetrue
10162     \pgf@relevantforpicturesizefalse
10163   }
10164   { \begin { pgfpicture } }
10165   \@@_qpoint:n { row - ##1 }
10166   \dim_set_eq:NN \l_tmpa_dim \pgf@y
10167   \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
10168   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
10169   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
10170   \bool_if:NTF \l_@@_in_code_after_bool
10171   { \endpgfpicture }
10172   { \end { pgfpicture } }
10173   \int_step_inline:nn \c@jCol
10174   {
10175     \hbox_set:Nn \l_tmpa_box
10176     {
10177       \normalfont \Large \sffamily \bfseries
10178       \bool_if:NTF \l_@@_in_code_after_bool
10179       { \color { red } }
10180       { \color { red ! 50 } }
10181       ##1 - ####1
10182     }
10183     \bool_if:NTF \l_@@_in_code_after_bool
10184     {
10185       \pgfpicture
10186       \pgfrememberpicturepositiononpagetrue
10187       \pgf@relevantforpicturesizefalse
10188     }
10189     { \begin { pgfpicture } }
10190     \@@_qpoint:n { col - ####1 }
10191     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
10192     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
10193     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
10194     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
10195     \bool_if:NTF \l_@@_in_code_after_bool
10196     { \endpgfpicture }
10197     { \end { pgfpicture } }
10198     \fp_set:Nn \l_tmpa_fp
10199     {
10200       \fp_min:nn
10201       {
10202         \fp_min:nn
10203         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
10204         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
10205       }
10206       { 1.0 }
10207     }
10208     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
10209     \pgfpicture
10210     \pgfrememberpicturepositiononpagetrue
10211     \pgf@relevantforpicturesizefalse
10212     \pgftransformshift
10213     {
10214       \pgfpoint
10215       { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
10216       \g_tmpa_dim

```

```

10217         }
10218         \pgfnode
10219         { rectangle }
10220         { center }
10221         { \box_use:N \l_tmpa_box }
10222         { }
10223         { }
10224         \endpgfpicture
10225     }
10226 }
10227 }

```

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.

```

10228 \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.

```

10229 \bool_new:N \g_@@_footnote_bool

10230 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
10231 {
10232     You-have-used-the-key~' \l_keys_key_str '~when-loading-nicematrix~
10233     but~that~key~is~unknown. \\\
10234     It~will~be~ignored. \\\
10235     For~a~list~of~the~available~keys,~type~H~<return>.
10236 }
10237 {
10238     The-available-keys-are~(in~alphabetic-order):~
10239     footnote,~
10240     footnotehyper,~
10241     messages-for-Overleaf,~
10242     renew-dots~and~
10243     renew-matrix.
10244 }

10245 \keys_define:nn { nicematrix }
10246 {
10247     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
10248     renew-dots .value_forbidden:n = true ,
10249     renew-matrix .code:n = \@@_renew_matrix: ,
10250     renew-matrix .value_forbidden:n = true ,
10251     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
10252     footnote .bool_set:N = \g_@@_footnote_bool ,
10253     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
10254     unknown .code:n = \@@_error:n { Unknown~key~for~package }
10255 }
10256 \ProcessKeyOptions

```

```

10257 \@@_msg_new:nn { footnote~with~footnotehyper~package }
10258 {
10259     You-can't-use-the-option~'footnote'~because~the~package~

```

```

10260 footnotehyper~has~already~been~loaded.~
10261 If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
10262 within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10263 of~the~package~footnotehyper.\\
10264 The~package~footnote~won't~be~loaded.
10265 }
10266 \@@_msg_new:nn { footnotehyper~with~footnote~package }
10267 {
10268   You~can't~use~the~option~'footnotehyper'~because~the~package~
10269   footnote~has~already~been~loaded.~
10270   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
10271   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10272   of~the~package~footnote.\\
10273   The~package~footnotehyper~won't~be~loaded.
10274 }
10275 \bool_if:NT \g_@@_footnote_bool
10276 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10277 \IfClassLoadedTF { beamer }
10278 { \bool_set_false:N \g_@@_footnote_bool }
10279 {
10280   \IfPackageLoadedTF { footnotehyper }
10281   { \@@_error:n { footnote~with~footnotehyper~package } }
10282   { \usepackage { footnote } }
10283 }
10284 }
10285 \bool_if:NT \g_@@_footnotehyper_bool
10286 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10287 \IfClassLoadedTF { beamer }
10288 { \bool_set_false:N \g_@@_footnote_bool }
10289 {
10290   \IfPackageLoadedTF { footnote }
10291   { \@@_error:n { footnotehyper~with~footnote~package } }
10292   { \usepackage { footnotehyper } }
10293 }
10294 \bool_set_true:N \g_@@_footnote_bool
10295 }

```

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.

```

10296 \bool_new:N \l_@@_underscore_loaded_bool
10297 \IfPackageLoadedT { underscore }
10298 { \bool_set_true:N \l_@@_underscore_loaded_bool }
10299 \AtBeginDocument
10300 {
10301   \bool_if:NF \l_@@_underscore_loaded_bool
10302   {

```

```

10303     \IfPackageLoadedT { underscore }
10304     { \@@_error:n { underscore~after~nicematrix } }
10305   }
10306 }

```

41 Compatibility with threeparttable

```

10307 \AtBeginDocument
10308 {
10309   \IfPackageLoadedT { threeparttable }
10310   {
10311     \AddToHook { env / threeparttable / begin }
10312     {
10313       \TPT@hookin { NiceTabular }
10314       \TPT@hookin { NiceTabular* }
10315       \TPT@hookin { NiceTabularX }
10316     }
10317   }
10318 }

```

42 Error messages of the package

When there is a unknown key, maybe the user has tried to use an inexistent “additive syntax” for that key. Of course, in that case, the last character of the name of the key is +.

#1 is a clist of names of sets of keys and #2 is the error message to send.

```

10319 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
10320 {
10321   \str_if_eq:eeTF
10322   { \str_item:Nn \l_keys_key_str { \str_count:N \l_keys_key_str } }
10323   { + }
10324   {
10325     \str_set:Ne \l_tmpa_str
10326     { \str_range:Nnn \l_keys_key_str { 1 } { \str_count:N \l_keys_key_str - 1 } }
10327     \bool_set_false:N \l_tmpa_bool
10328     \clist_map_inline:nn { #1 }
10329     {
10330       \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10331       {
10332         \@@_error:n { key~without~+~exists }
10333         \bool_set_true:N \l_tmpa_bool
10334         \clist_map_break:
10335       }
10336     }
10337     \bool_if:NF \l_tmpa_bool
10338     {
10339       \str_set:Ne \l_keys_key_str { \tl_trim_right_spaces:V \l_tmpa_str }
10340       \@@_unknown_key_i:nn { #1 } { #2 }
10341     }
10342   }
10343   { \@@_unknown_key_i:nn { #1 } { #2 } }
10344 }

```

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.

```

10345 \cs_new_protected:Npn \@@_unknown_key_i:nn #1 #2
10346 {
10347   \str_set_eq:NN \l_tmpa_str \l_keys_key_str
10348   \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
10349   \str_set:Ne \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
10350   \bool_set_false:N \l_tmpa_bool
10351   \clist_map_inline:nn { #1 }
10352   {
10353     \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10354     {
10355       \@@_error:n { key~with~normal~form~exists }
10356       \bool_set_true:N \l_tmpa_bool
10357       \clist_map_break:
10358     }
10359   }
10360   \bool_if:NF \l_tmpa_bool
10361   {
10362     \@@_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).

```

10363     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10364     { \msg_info:nn { nicematrix } { #2~+ } }
10365   }
10366 }

10367 \@@_msg_new:nn { key~without~+~exists }
10368 {
10369   The~key~'\tl_trim_right_spaces:V \l_tmpa_str'~exists~but~does~not~accept~an~
10370   additive~syntax~(with~+=).\
10371   It~will~be~ignored.\
10372 }

10373 \@@_msg_new:nn { key~with~normal~form~exists }
10374 {
10375   No~key~'\l_keys_key_str'.\
10376   It~will~be~ignored.\
10377   Maybe~you~want~to~use~the~key~'\l_tmpa_str'.
10378 }

10379 \str_const:Ne \c_@@_available_keys_str
10380 {
10381   \bool_if:nT { ! \g_@@_messages_for_Overleaf_bool }
10382   { For~a~list~of~the~available~keys,~type~H~<return>. }
10383 }

10384 \seq_new:N \g_@@_types_of_matrix_seq
10385 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
10386 {
10387   NiceMatrix ,
10388   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
10389 }
10390 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
10391 { \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`.

```

10392 \cs_new_protected:Npn \@@_err_too_many_cols:
10393 {
10394   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
10395   { \@@_fatal:nn { too-many-cols-for-array } }
10396   \int_compare:nNnT \l_@@_last_col_int = { -2 }
10397   { \@@_fatal:n { too-many-cols-for-matrix } }
10398   \int_compare:nNnT \l_@@_last_col_int = { -1 }
10399   { \@@_fatal:n { too-many-cols-for-matrix } }
10400   \bool_if:NF \l_@@_last_col_without_value_bool
10401   { \@@_fatal:n { too-many-cols-for-matrix-with-last-col } }
10402 }

```

The following command must *not* be protected since it's used in an error message.

```

10403 \cs_new:Npn \@@_message_hdotsfor:
10404 {
10405   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
10406   { ~Maybe-your-use-of~ \token_to_str:N \Hdotsfor \ or~
10407     \token_to_str:N \Hbrace \ is-incorrect. }
10408 }

10409 \cs_new_protected:Npn \@@_Hline_in_cell:
10410 { \@@_fatal:n { Misuse-of-Hline } }

10411 \@@_msg_new:nn { Misuse-of-Hline }
10412 {
10413   Misuse-of-Hline. \\
10414   Error-in-your-row~ \int_eval:N \c@iRow . \\
10415   \token_to_str:N \Hline\ (like \token_to_str:N \hline)~must-be-used-only-
10416   at-the-beginning-of-a-row.\\
10417   That-error-is-fatal.
10418 }

10419 \@@_msg_new:nn { hvlines,~rounded-corners-and-corners }
10420 {
10421   Incompatible-options.\\
10422   You-should-not-use-'hvlines',~'rounded-corners'~and-'corners'~at-the-same-time.\\
10423   The-output-will-not-be-reliable.
10424 }

10425 \@@_msg_new:nn { Body-alone }
10426 {
10427   \token_to_str:N \Body\ alone. \\
10428   You-have-used~\token_to_str:N \Body\ without~\token_to_str:N \CodeBefore.\\
10429   That-error-is-fatal.
10430 }

10431 \@@_msg_new:nn { Bad-use-of-CodeBefore }
10432 {
10433   Bad-use-of~\token_to_str:N \CodeBefore. \\
10434   \token_to_str:N \CodeBefore\ must-be-used-only-at-the-beginning-of-
10435   the-environment.\\
10436   That-error-is-fatal.
10437 }

10438 \@@_msg_new:nn { NiceTabularX-probably-required }
10439 {
10440   Incorrect-syntax.\\
10441   You-probably-want-to-use~\{NiceTabularX\}~whose-first-argument-is-the-
10442   disered-width-of-the-tabular.\\
10443   That-error-is-fatal.
10444 }

10445 \@@_msg_new:nn { cellcolor-in-Block }
10446 {
10447   Bad-use-of~\token_to_str:N \cellcolor \\
10448   You-can't-use~\token_to_str:N \cellcolor\ in~\token_to_str:N \Block\
10449   \bool_if:NTF \l_@@_amp_in_blocks_bool

```



```

10450     { (but-you-could-use-it-in-a-sub-block-since-'\&-in-blocks'-is-in-force) }
10451     { (it's-possible-in-a-sub-block-when-'\&-in-blocks'-is-in-force) }
10452     .~Here,~you-should-use-the-key~'fill'~of~the~block.\\
10453     That~command-will-be-ignored.
10454 }
10455 \@@_msg_new:nn { rowcolor~in~Block }
10456 {
10457     Bad-use-of~\token_to_str:N \rowcolor \\
10458     You-can't-use~\token_to_str:N \rowcolor\ in~\token_to_str:N \Block.\\
10459     However,~it's-possible-to-color-the-block-with-its-key~'fill'.\\
10460     That~command-will-be-ignored.
10461 }
10462 \@@_msg_new:nn { key~color~inside }
10463 {
10464     Deleted~key.\\
10465     The-key~'color~inside'~(and-its-alias~'colortbl-like')~has-been-deleted-in
10466     ~'nicematrix'~and-must-not-be-used.\\
10467     This-error-is-fatal.
10468 }
10469 \@@_msg_new:nn { Invalid~argument~for~w }
10470 {
10471     Invalid~argument~for~w.\\
10472     You-have-used-the-type-of~alignment~'#1'~for~your~column~'w'~
10473     but-only~'c',~'r',~'l'~and~'s'~are-allowed-in-a-column~'w'.~
10474     If-you-go-on,~'c'~will-be-used.
10475 }
10476 \@@_msg_new:nn { invalid~weight }
10477 {
10478     Unknown~key.\\
10479     The-key~'\l_keys_key_str'~of~your~column~X~is-unknown-and-will-be-ignored.~
10480     The-available-keys-are:~l,~c,~r,~t(=p),~m,~b,~V~
10481     \IfPackageLoadedTF { varwidth }
10482     { (since~'varwidth'~is-loaded)~}
10483     { (if-you-load~'varwidth')~}
10484     and-real-numbers-for~the-weight-of~the~X~column.
10485 }
10486 \@@_msg_new:nn { last~col~not~used }
10487 {
10488     Column~not~used.\\
10489     The-key~'last~col'~is-in-force-but-you-have-not-used-that-last-column~
10490     in~your~\@@_full_name_env: .~
10491     However,~you-can-go-on.
10492 }
10493 \@@_msg_new:nn { too-many~cols~for~matrix~with~last~col }
10494 {
10495     Too-many-columns.\\
10496     In~the~row~ \int_eval:n { \c@iRow },~
10497     you-try-to-use-more-columns~
10498     than-allowed-by-your~ \@@_full_name_env: .
10499     \@@_message_hdotsfor: \
10500     The-maximal-number-of-columns-is~ \int_eval:n { \l_@@_last_col_int - 1 }~
10501     (plus~the~exterior~columns).\\
10502     But,~maybe,~you-have-forgotten-a~\token_to_str:N \\. \\
10503     This-error-is-fatal.
10504 }
10505 \@@_msg_new:nn { too-many~cols~for~matrix }
10506 {
10507     Too-many-columns.\\
10508     In~the~row~ \int_eval:n { \c@iRow },~
10509     you-try-to-use-more-columns~than-allowed-by-your~ \@@_full_name_env: .
10510     \@@_message_hdotsfor: \

```

```

10511 Recall~that~the~maximal~number~of~columns~for~a~matrix~
10512 (excepted~the~potential~exterior~columns)~is~fixed~by~the~
10513 LaTeX~counter~'MaxMatrixCols'.~
10514 Its~current~value~is~ \int_use:N \c@MaxMatrixCols \
10515 (use~ \token_to_str:N \setcounter \ to~change~that~value).\
10516 But,~maybe,~you~have~forgotten~a~\token_to_str:N \\. \
10517 This~error~is~fatal.
10518 }

10519 \@@_msg_new:nn { too-many-cols-for-array }
10520 {
10521   Too-many-columns.\
10522   In~the~row~ \int_eval:n { \c@iRow } ,~
10523   ~you~try~to~use~more~columns~than~allowed~by~your~
10524   \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal~number~of~columns~is~
10525   \int_use:N \g_@@_static_num_of_col_int \
10526   \bool_if:nT
10527     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
10528     { (plus~the~exterior~ones)~}
10529   since~the~preamble~is~' \g_@@_user_preamble_tl '.\
10530   But,~maybe,~you~have~forgotten~a~\token_to_str:N \\. \
10531   This~error~is~fatal.
10532 }

10533 \@@_msg_new:nn { columns-not-used }
10534 {
10535   Columns-not-used.\
10536   The~preamble~of~your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '.~
10537   It~announces~ \int_use:N \g_@@_static_num_of_col_int \
10538   columns~but~you~only~used~ \int_use:N \c@jCol .\
10539   The~columns~you~did~not~used~won't~be~created.\
10540   You~won't~have~similar~warning~till~the~end~of~the~document.
10541 }

10542 }
10543 \@@_msg_new:nn { Bad-use-of-NiceTabularNotes }
10544 {
10545   Bad-use-of~\token_to_str:N \NiceTabularNotes \
10546   \token_to_str:N \NiceTabularNotes\ should~be~used~only~
10547   after~at~tabular~which~uses~`notes/no-print`. \
10548   That~command~will~be~ignored.
10549 }

10550 \@@_msg_new:nn { empty-preamble }
10551 {
10552   Empty-preamble.\
10553   The~preamble~of~your~ \@@_full_name_env: \ is~empty.\
10554   This~error~is~fatal.
10555 }

10556 \@@_msg_new:nn { in-first-col }
10557 {
10558   Erroneous-use.\
10559   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\
10560   That~command~will~be~ignored.\
10561   You~can~try~to~delete~the~key~'first-col'.
10562 }

10563 \@@_msg_new:nn { in-last-col }
10564 {
10565   Erroneous-use.\
10566   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\
10567   That~command~will~be~ignored.\
10568   You~can~try~to~delete~the~key~'last-col'.
10569 }

```

```

10570 \@@_msg_new:nn { in~first~row }
10571 {
10572   Erroneous~use.\\
10573   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
10574   That~command~will~be~ignored.\\
10575   You~can~try~to~delete~the~key~'first~row'.
10576 }
10577 \@@_msg_new:nn { in~last~row }
10578 {
10579   Erroneous~use.\\
10580   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
10581   That~command~will~be~ignored.\\
10582   You~can~try~to~delete~the~key~'last~row'.
10583 }
10584 \@@_msg_new:nn { TopRule~without~booktabs }
10585 {
10586   Erroneous~use.\\
10587   You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
10588   You~should~load~'booktabs'~(before~or~after~'nicematrix').\\
10589   That~command~will~be~ignored.
10590 }
10591 \@@_msg_new:nn{ rotate~in~p~col }
10592 {
10593   \token_to_str:N \rotate\ forbidden.\\
10594   You~should~not~use~\token_to_str:N \rotate\ in~a~column~of~type~'p',~
10595   'b',~'m'\IfPackageLoadedTF { varwidth } { ,~'X'~or~'V' } { ~or~'X'}.~
10596   If~you~go~on,~maybe~you~won't~have~the~expected~output.~You~should~
10597   consider~the~classical~command~\token_to_str:N \rotatebox.
10598 }
10599 \@@_msg_new:nn { TopRule~without~tikz }
10600 {
10601   Erroneous~use.\\
10602   You~can't~use~the~command~ #1 because~TikZ~is~not~loaded.\\
10603   You~should~load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.\\
10604   \IfPackageLoadedF { booktabs }
10605   { You~should~also~load~'booktabs'.\\ }
10606   That~command~will~be~ignored.
10607 }
10608 \@@_msg_new:nn { caption~outside~float }
10609 {
10610   Key~caption~forbidden.\\
10611   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
10612   environment~(such~as~\{table\}).~This~key~will~be~ignored.
10613 }
10614 \@@_msg_new:nn { short~caption~without~caption }
10615 {
10616   You~should~not~use~the~key~'short~caption'~without~'caption'.~
10617   However,~your~'short~caption'~will~be~used~as~'caption'.
10618 }
10619 \@@_msg_new:nn { double~closing~delimiter }
10620 {
10621   Double~delimiter.\\
10622   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
10623   delimiter.~This~delimiter~will~be~ignored.\\
10624   You~can~try~to~use~\token_to_str:N \SubMatrix\ in~the~\token_to_str:N \CodeAfter.
10625 }
10626 \@@_msg_new:nn { delimiter~after~opening }
10627 {
10628   Double~delimiter.\\
10629   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~

```

```

10630     delimiter.~That~delimiter~will~be~ignored.
10631 }
10632 \@@_msg_new:nn { bad~option~for~line~style }
10633 {
10634     Bad~line~style.\\
10635     Since~you~haven't~loaded~TikZ,~the~only~value~you~can~give~to~'line~style'~
10636     is~'standard'.~That~key~will~be~ignored.\\
10637     You~can~load~TikZ~with~\token_to_str:N \usepackage \{tikz\},~
10638     before~or~after~'nicematrix'.\\
10639 }
10640 \@@_msg_new:nn { draw~trees~with~no~cell~nodes }
10641 {
10642     Incompatible~keys.\\
10643     You~can't~use~the~key~'draw~trees~in~col'~here~because~the~key~'no~cell~nodes'~
10644     is~in~force~(you~should~deactive~the~key~'no~cell~nodes'~whose~only~goal~
10645     is~to~speed~compilation~up).\\
10646     If~you~go~on,~that~key~will~be~ignored.
10647 }
10648 \@@_msg_new:nn { corners~with~no~cell~nodes }
10649 {
10650     Incompatible~keys.\\
10651     You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
10652     is~in~force~(you~should~deactive~the~key~'no~cell~nodes'~whose~only~goal~
10653     is~to~speed~compilation~up).\\
10654     If~you~go~on,~that~key~will~be~ignored.
10655 }
10656 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
10657 {
10658     Incompatible~keys.\\
10659     You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
10660     is~in~force~(you~should~deactive~the~key~'no~cell~nodes'~whose~only~goal~
10661     is~to~speed~compilation~up).\\
10662     If~you~go~on,~those~extra~nodes~won't~be~created.
10663 }
10664 \@@_msg_new:nn { Identical~notes~in~caption }
10665 {
10666     Identical~tabular~notes.\\
10667     You~can't~put~several~notes~with~the~same~content~in~
10668     \token_to_str:N \caption \ (but~it's~possible~in~the~main~tabular).\\
10669     If~you~go~on,~the~output~will~probably~be~erroneous.
10670 }
10671 \@@_msg_new:nn { tabularnote~below~the~tabular }
10672 {
10673     \token_to_str:N \tabularnote \ forbidden\\
10674     You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
10675     of~your~tabular~because~the~caption~will~be~composed~below~
10676     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
10677     key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .\\
10678     Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
10679     no~similar~error~will~raised~in~this~document.
10680 }
10681 \@@_msg_new:nn { Unknown~key~for~rules }
10682 {
10683     Unknown~key.\\
10684     There~are~only~three~keys~available~here:~'width',~'color'~and~
10685     'fix~vertex'.\\
10686     Your~key~' \l_keys_key_str ' ~will~be~ignored.
10687 }
10688 \@@_msg_new:nn { Unknown~key~for~trees }
10689 {

```

```

10690     Unknown~key.\\
10691     There~are~only~three~keys~available~here:~width~color~and~
10692     rounded~corners.\\
10693     Your~key~' \l_keys_key_str '~will~be~ignored.
10694 }
10695 % \end{macrocode}
10696 %
10697 %
10698 % \begin{macrocode}
10699 \@@_msg_new:nn { Unknown~key~for~Hbrace }
10700 {
10701     Unknown~key.\\
10702     You~have~used~the~key~' \l_keys_key_str '~but~the~only~
10703     keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
10704     and~ \token_to_str:N \Vbrace \ are:~'brace-shift(+)',~'color',~
10705     'horizontal-label(s)',~'shorten'~'shorten-end'~
10706     and~'shorten-start'.\\
10707     That~error~is~fatal.
10708 }
10709 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
10710 {
10711     Unknown~key.\\
10712     There~is~only~two~keys~available~here:~
10713     'empty'~and~'not-empty'.\\
10714     Your~key~' \l_keys_key_str '~will~be~ignored.
10715 }
10716 \@@_msg_new:nn { Unknown~key~for~rotate }
10717 {
10718     Unknown~key.\\
10719     The~only~keys~available~here~are~'c'~and~'-90'.\\
10720     Your~key~' \l_keys_key_str '~will~be~ignored.
10721 }
10722 \@@_msg_new:nnn { Unknown~key~for~custom~line }
10723 {
10724     Unknown~key.\\
10725     The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
10726     It~you~go~on,~you~will~probably~have~other~errors. \\
10727     \c_@@_available_keys_str
10728 }
10729 {
10730     The~available~keys~are~(in~alphabetic~order):~
10731     ccommand,~
10732     color,~
10733     command,~
10734     dotted,~
10735     letter,~
10736     multiplicity,~
10737     sep-color,~
10738     tikz,~and~total-width.
10739 }
10740 \@@_msg_new:nnn { Unknown~key~for~default~line }
10741 {
10742     Unknown~key.\\
10743     The~key~' \l_keys_key_str '~is~unknown~in~a~'default~line'.~
10744     It~you~go~on,~you~will~probably~have~other~errors. \\
10745     \c_@@_available_keys_str
10746 }
10747 {
10748     The~available~keys~are~(in~alphabetic~order):~
10749     color,~
10750     dotted,~
10751     multiplicity,~

```

```

10752     sep-color,~
10753     tikz,~and~total-width.
10754 }

10755 \@@_msg_new:nnn { Unknown~key~for~xdots }
10756 {
10757     Unknown~key.\\
10758     The~key~' \l_keys_key_str '-is~unknown~for~a~command~for~drawing~dotted~rules.\\
10759     \c_@@_available_keys_str
10760 }
10761 {
10762     The~available~keys~are~(in~alphabetic~order):~
10763     'color',~
10764     'horizontal(s)-labels',~
10765     'inter',~
10766     'line-style',~
10767     'nullify',~
10768     'radius',~
10769     'shorten',~
10770     'shorten-end'~and~'shorten-start'.
10771 }

10772 \@@_msg_new:nn { Unknown~key~for~rowcolors }
10773 {
10774     Unknown~key.\\
10775     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
10776     (and~you~try~to~use~' \l_keys_key_str ')\
10777     That~key~will~be~ignored.
10778 }

10779 \@@_msg_new:nn { Col~outside~tabular~in~trees }
10780 {
10781     Error~with~'draw-trees-in-col' \\
10782     The~number~of~column~'#1'~is~outside~your~tabular~since~the~last~column~
10783     is~\int_use:N \c@jCol. \\
10784     It~will~be~ignored.
10785 }

10786 \@@_msg_new:nn { Last~col~in~trees }
10787 {
10788     Error~with~'draw-trees-in-col' \\
10789     You~can't~use~'draw-trees-in-col'~with~the~column~'#1'~
10790     because~it's~the~last~column~of~your~tabular. \\
10791     It~will~be~ignored.
10792 }

10793 \@@_msg_new:nn { label~without~caption }
10794 {
10795     You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
10796     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
10797 }

10798 \@@_msg_new:nn { W~warning }
10799 {
10800     Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
10801     (row~ \int_use:N \c@iRow ).
10802 }

10803 \@@_msg_new:nn { Construct~too~large }
10804 {
10805     Construct~too~large.\\
10806     Your~command~ \token_to_str:N #1
10807     can't~be~drawn~because~your~matrix~is~too~small.\\
10808     That~command~will~be~ignored.
10809 }

10810 \@@_msg_new:nn { underscore~after~nicematrix }
10811 {

```

```

10812 Problem-with-'underscore'.\\
10813 The-package-'underscore'~should-be-loaded-before-'nicematrix'.~
10814 You-can-go-on-but-you-won't-be-able-to-write-something-such-as:\\
10815 ' \token_to_str:N \Cdots \token_to_str:N _
10816 \{ n \token_to_str:N \text \{ ~times \} \}' .
10817 }
10818 \@@_msg_new:nn { ampersand-in-light-syntax }
10819 {
10820 Ampersand~forbidden.\\
10821 You-can't-use-an-ampersand~( \token_to_str:N &)~to-separate-columns~because~
10822 ~the-key-'light-syntax'~is-in-force.~This-error-is-fatal.
10823 }
10824 \@@_msg_new:nn { double-backslash-in-light-syntax }
10825 {
10826 Double~backslash~forbidden.\\
10827 You-can't-use~ \token_to_str:N \\
10828 ~to-separate-rows~because~the-key-'light-syntax'~
10829 is-in-force.~You-must-use~the-character~' \l_@@_end_of_row_tl '~
10830 (set~by~the-key-'end-of-row').~This-error-is-fatal.
10831 }
10832 \@@_msg_new:nn { hlines-with-color }
10833 {
10834 Incompatible-keys.\\
10835 You-can't-use~the-keys-'hlines',~'vlines'~or~'hvlines'~for~a~
10836 \token_to_str:N \Block \ when~the-key-'color'~or~'draw'~is-used.\\
10837 However,~you-can-put~several-commands~ \token_to_str:N \Block.\\
10838 Your-key-will-be-discarded.
10839 }
10840 \@@_msg_new:nn { bad-value-for-baseline }
10841 {
10842 Bad-value-for-baseline.\\
10843 The-value-given-to-'baseline'~( \int_use:N \l_tmpa_int )~is-not~
10844 valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int\ and~
10845 \int_use:N \g_@@_row_total_int \ or~equal-to~'t',~'c'~or~'b'~or~of~
10846 the-form-'line-i'.\\
10847 A-value-of~1~will-be-used.
10848 }
10849 \@@_msg_new:nn { bad-value-for-baseline-line }
10850 {
10851 Bad-value-for-baseline-with-line.\\
10852 The-value-given-to-'baseline'~( \int_use:N \l_tmpa_int )~is-not~
10853 valid.~The-number-of~the-line-must-be-between~1~and~
10854 \int_eval:n { \c@iRow + 1 } \\
10855 A-value-of~'line-1'~will-be-used.
10856 }
10857 \@@_msg_new:nn { detection-of-empty-cells }
10858 {
10859 Problem-with-'not-empty'\\
10860 For-technical-reasons,~you-must-activate~
10861 'create-cell-nodes'~in~ \token_to_str:N \CodeBefore \
10862 in-order-to-use~the-key~' \l_keys_key_str ' .\\
10863 That-key-will-be-ignored.
10864 }
10865 \@@_msg_new:nn { siunitx-too-old }
10866 {
10867 siunitx-too-old\\
10868 You-can't-use~the-columns-'S'~because~your-version-of-'siunitx'~
10869 is-too-old.~You-need-at-least~the-version~3.5.1~(2026/03/26).\\
10870 That-error-is-fatal.\\
10871 }

```

```

10872 \@@_msg_new:nn { Invalid-name }
10873 {
10874   Invalid-name.\
10875   You-can't-give-the-name~' \l_keys_value_tl '~to-a~ \token_to_str:N
10876   \SubMatrix \ of~your~ \@@_full_name_env: .\
10877   A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\
10878   This-key-will-be-ignored.
10879 }

10880 \@@_msg_new:nn { Hbrace-not-allowed }
10881 {
10882   Command-not-allowed.\
10883   You-can't-use-the-command~ \token_to_str:N #1
10884   because-you-have-not-loaded~
10885   \IfPackageLoadedTF { tikz }
10886     { the-TikZ-library~'decorations.pathreplacing'~Use~ }
10887     { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
10888   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \
10889   That-command-will-be-ignored.
10890 }

10891 \@@_msg_new:nn { Vbrace-not-allowed }
10892 {
10893   Command-not-allowed.\
10894   You-can't-use-the-command~ \token_to_str:N \Vbrace \
10895   because-you-have-not-loaded-TikZ~
10896   and-the-TikZ-library~'decorations.pathreplacing'~.\
10897   Use: ~\token_to_str:N \usepackage \{tikz\}~
10898   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \
10899   That-command-will-be-ignored.
10900 }

10901 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
10902 {
10903   Wrong-line.\
10904   You-try-to-draw-a-#1~line-of-number~'#2'~in-a~
10905   \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but-that~
10906   number-is-not-valid.~It-will-be-ignored.
10907 }

10908 \@@_msg_new:nn { Impossible-SubMatrix }
10909 {
10910   Impossible-SubMatrix.\
10911   It's-impossible-to-draw~your~\token_to_str:N \SubMatrix \
10912   because-all-the-cells-are-empty-in-the-column-on-the-#1~
10913   side-of-that~\token_to_str:N \SubMatrix.
10914   \bool_if:NT \l_@@_submatrix_slim_bool
10915     { ~Maybe-you-should-try-without-the-key~'slim'. } \
10916   This~ \token_to_str:N \SubMatrix \ will-be-ignored.
10917 }

10918 \@@_msg_new:nn { Impossible-SubMatrix-no-cell-nodes }
10919 {
10920   Impossible-SubMatrix.\
10921   It's-impossible-to-draw~your~ \token_to_str:N \SubMatrix \
10922   because-the-key~'no-cell-nodes'~is-in-force. \
10923   This~ \token_to_str:N \SubMatrix \ will-be-ignored.
10924 }

10925 \@@_msg_new:nnn { width-without-X-columns }
10926 {
10927   You-have-used-the-key~'width'~but-you-have-put-no~'X'~column-in~
10928   the-preamble~(' \g_@@_user_preamble_tl '~)~of~your~ \@@_full_name_env: .\
10929   That-key-will-be-ignored.
10930 }
10931 {
10932   This-message-is-the-message~'width-without-X-columns'~

```



```

10933     of~the~module~'nicematrix'.~
10934     The~experimented~users~can~disable~that~message~with~
10935     \token_to_str:N \msg_redirect_name:nnn .\\
10936 }
10937
10938 \@@_msg_new:nn { key-multiplicity-with-dotted }
10939 {
10940     Incompatible-keys. \\
10941     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
10942     in~a~'custom-line'.~They~are~incompatible. \\
10943     The~key~'multiplicity'~will~be~discarded.
10944 }
10945 \@@_msg_new:nn { empty-environment }
10946 {
10947     Empty~environment.\\
10948     Your~ \@@_full_name_env: \ is~empty.~This~error~is~fatal.
10949 }
10950 \@@_msg_new:nn { No~letter~and~no~command }
10951 {
10952     Erroneous~use.\\
10953     Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
10954     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10955     ~'ccommand'~(to~draw~horizontal~rules).\\
10956     However,~you~can~go~on.
10957 }
10958 \@@_msg_new:nn { Forbidden~letter }
10959 {
10960     Forbidden~letter.\\
10961     You~can't~use~the~letter~'#1'~for~a~customized~line.~
10962     It~will~be~ignored.\\
10963     The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10964 }
10965 \@@_msg_new:nn { Several~letters }
10966 {
10967     Wrong~name.\\
10968     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10969     have~used~' \l_@@_letter_str ').\\
10970     It~will~be~ignored.
10971 }
10972 \@@_msg_new:nn { Delimiter~with~small }
10973 {
10974     Delimiter~forbidden.\\
10975     You~can't~put~a~delimiter~in~the~preamble~of~your~
10976     \@@_full_name_env: \
10977     because~the~key~'small'~is~in~force.\\
10978     This~error~is~fatal.
10979 }
10980 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10981 {
10982     Unknown~cell.\\
10983     Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10984     the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10985     can't~be~executed~because~a~cell~doesn't~exist.\\
10986     This~command~ \token_to_str:N \line \ will~be~ignored.
10987 }
10988 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10989 {
10990     Duplicate~name.\\
10991     The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10992     in~this~ \@@_full_name_env: .\\

```

```

10993     This-key-will-be-ignored.\\
10994     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10995     { For-a-list-of-the-names-already-used,~type-H<return>. }
10996   }
10997   {
10998     The-names-already-defined-in-this~ \@@_full_name_env: \ are:~
10999     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
11000   }
11001   \@@_msg_new:nn { r-or-l-with-preamble }
11002   {
11003     Erroneous-use.\\
11004     You-can't-use-the-key~' \l_keys_key_str '~in-your~ \@@_full_name_env: .~
11005     You-must-specify-the-alignment-of-your-columns-with-the-preamble-of~
11006     your~ \@@_full_name_env: .\\
11007     This-key-will-be-ignored.
11008   }
11009   \@@_msg_new:nn { Hdotsfor-in-col-0 }
11010   {
11011     Erroneous-use.\\
11012     You-can't-use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\
11013     in-an-exterior-column-of~
11014     the-array.~This-error-is-fatal.
11015   }
11016   \@@_msg_new:nn { bad-corner }
11017   {
11018     Bad-corner.\\
11019     '#1'~is-an-incorrect-specification-for-a-corner~(in-the-key~
11020     'corners').~The-available-values-are:~NW,~SW,~NE-and~SE.\\
11021     This-specification-of-corner-will-be-ignored.
11022   }
11023   \@@_msg_new:nn { bad-border }
11024   {
11025     Bad-border.\\
11026     '\l_keys_key_str'~is-an-incorrect-specification-for-a-border~
11027     (in-the-key~'borders'~of-the-command~ \token_to_str:N \Block ).~
11028     The-available-values-are:~left,~right,~top-and-bottom~(and-you-can~
11029     also-use-the-key~'tikz'
11030     \IfPackageLoadedF { tikz }
11031     { ~if-you-load-the-LaTeX-package~'tikz' } ).\\
11032     This-specification-of-border-will-be-ignored.
11033   }
11034   \@@_msg_new:nn { TikzEveryCell-without-tikz }
11035   {
11036     TikZ-not-loaded.\\
11037     You-can't-use~ \token_to_str:N \TikzEveryCell \
11038     because-you-have-not-loaded-TikZ.\\
11039     You-can-load-TikZ-with~\token_to_str:N \usepackage \{tikz\},~
11040     before~or~after~'nicematrix'. \\
11041     This-command-will-be-ignored.
11042   }
11043   \@@_msg_new:nn { tikz-key-without-tikz }
11044   {
11045     TikZ-not-loaded.\\
11046     You-can't-use-the-key~'tikz'~for-the-command~' \token_to_str:N
11047     \Block '~because-you-have-not-loaded-TikZ.\\
11048     You-can-load-TikZ-with~\token_to_str:N \usepackage \{tikz\},~
11049     before~or~after~'nicematrix'. \\
11050     This-key-will-be-ignored.
11051   }
11052   \@@_msg_new:nn { Bad-argument-for-Block }
11053   {

```

```

11054     Bad-argument.\\
11055     The-first-mandatory-argument-of~\token_to_str:N \Block\ must~
11056     be-of-the-form~'i-j'~(or-totally-empty)~and-you-have-used:~
11057     '#1'. \\
11058     If-you-go-on,~the~\token_to_str:N \Block\ will-be-mono-cell~(as-if~
11059     the-argument-was-empty).
11060 }

11061 \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
11062 {
11063     Erroneous-use.\\
11064     In-the~ \@@_full_name_env: ,~you-must-use-the-key~
11065     'last-col'~without-value.\\
11066     However,~you-can-go-on-for-this-time~
11067     (the-value~' \l_keys_value_tl '~will-be-ignored).
11068 }

11069 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
11070 {
11071     Erroneous-use. \\
11072     In~\token_to_str:N \NiceMatrixOptions ,~you-must-use-the-key~
11073     'last-col'~without-value. \\
11074     However,~you-can-go-on-for-this-time~
11075     (the-value~' \l_keys_value_tl '~will-be-ignored).
11076 }

11077 \@@_msg_new:nn { Block-too-large-1 }
11078 {
11079     Block-too-large. \\
11080     You-try-to-draw-a-block-in-the-cell~#1-#2-of-your-matrix-but-the-matrix-is~
11081     too-small-for-that-block. \\
11082     This-block-and-maybe-others-will-be-ignored.
11083 }

11084 \@@_msg_new:nn { Block-too-large-2 }
11085 {
11086     Block-too-large. \\
11087     The-preamble-of-your~ \@@_full_name_env: \ announces~ \int_use:N
11088     \g_@@_static_num_of_col_int \
11089     columns-but-you-use-only~ \int_use:N \c@jCol \ and-that's-why-a-block~
11090     specified-in-the-cell~#1-#2-can't-be-drawn.~You-should-add-some-ampersands~
11091     (&)~at-the-end-of-the-first-row-of-your~ \@@_full_name_env: . \\
11092     This-block-and-maybe-others-will-be-ignored.
11093 }

11094 \@@_msg_new:nn { unknown-column-type }
11095 {
11096     Bad-column-type. \\
11097     The-column-type~'#1'~in-your~ \@@_full_name_env: \
11098     is-unknown. \\
11099     This-error-is-fatal.
11100 }

11101 \@@_msg_new:nn { unknown-column-type-multicolumn }
11102 {
11103     Bad-column-type. \\
11104     The-column-type~'#1'~in-the-command~\token_to_str:N \multicolumn \
11105     ~of-your~ \@@_full_name_env: \
11106     is-unknown. \\
11107     This-error-is-fatal.
11108 }

11109 \@@_msg_new:nn { unknown-column-type-S }
11110 {
11111     Bad-column-type. \\
11112     The-column-type~'S'~in-your~ \@@_full_name_env: \ is-unknown. \\
11113     If-you-want-to-use-the-column-type~'S'~of~siunitx,~you-should~
11114     load-that-package. \\

```

```

11115     This-error-is-fatal.
11116 }

11117 \@@_msg_new:nn { unknown-column-type-S-multicolumn }
11118 {
11119     Bad-column-type. \\
11120     The-column-type-'S'-in-the-command-\token_to_str:N \multicolumn \
11121     of-your- \@@_full_name_env: \ is-unknown. \\
11122     If-you-want-to-use-the-column-type-'S'-of-siunitx,~you-should-
11123     load-that-package. \\
11124     This-error-is-fatal.
11125 }

11126 \@@_msg_new:nn { tabularnote-forbidden }
11127 {
11128     Forbidden-command. \\
11129     You-can't-use-the-command- \token_to_str:N \tabularnote \
11130     ~here.~This-command-is-available-only-in-
11131     \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in-
11132     the-argument-of-a-command-\token_to_str:N \caption \ included-
11133     in-an-environment-\{table\}. \\
11134     This-command-will-be-ignored.
11135 }

11136 \@@_msg_new:nn { borders-forbidden }
11137 {
11138     Forbidden-key.\\
11139     You-can't-use-the-key-'borders'-of-the-command- \token_to_str:N \Block \
11140     because-the-option-'rounded-corners'-
11141     is-in-force-with-a-non-zero-value.\\
11142     This-key-will-be-ignored.
11143 }

11144 \@@_msg_new:nn { bottomrule-without-booktabs }
11145 {
11146     booktabs-not-loaded.\\
11147     You-can't-use-the-key-'tabular/bottomrule'~because-you-haven't-
11148     loaded-'booktabs'.~You-should-load-'booktabs',~before-or-
11149     after-'nicematrix'.\\
11150     This-key-will-be-ignored.
11151 }

11152 \@@_msg_new:nn { enumitem-not-loaded }
11153 {
11154     enumitem-not-loaded. \\
11155     You-can't-use-the-command- \token_to_str:N \tabularnote \
11156     ~because-you-haven't-loaded-'enumitem'.~We-should-load-it-
11157     (before-or-after-'nicematrix').\\
11158     All-the-commands- \token_to_str:N \tabularnote \ will-be-
11159     ignored-in-the-document.
11160 }

11161 \@@_msg_new:nn { tikz-without-tikz }
11162 {
11163     TikZ-not-loaded. \\
11164     You-can't-use-the-key-'tikz'~here~because~TikZ-is-not-
11165     loaded.~If-you-go-on,~that-key-will-be-ignored.
11166 }

11167 \@@_msg_new:nn { tikz-in-custom-line-without-tikz }
11168 {
11169     TikZ-not-loaded. \\
11170     You-have-used-the-key-'tikz'~in-the-definition-of-a-
11171     customized-line~(with-'custom-line')~but~TikZ-is-not-loaded.~
11172     You-can-go-on-but-you-will-have-another-error-if-you-actually-
11173     use-that-custom-line.
11174 }

```

```

11175 \@@_msg_new:nn { tikz-in-borders-without-tikz }
11176 {
11177   TikZ~not~loaded. \\
11178   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
11179   command~' \token_to_str:N \Block ')~but~TikZ~is~not~loaded.~
11180   That~key~will~be~ignored.
11181 }
11182 \@@_msg_new:nn { color-in-custom-line-with-tikz }
11183 {
11184   Erroneous~use.\\
11185   In~a~'custom-line',~you~have~used~both~'tikz'~and~'color',~
11186   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
11187   The~key~'color'~will~be~discarded.
11188 }
11189 \@@_msg_new:nn { Wrong-last-row }
11190 {
11191   Wrong~number.\\
11192   You~have~used~'last-row= \int_use:N \l_@@_last_row_int '~but~your~
11193   \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
11194   If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
11195   last~row~but~you~should~correct~your~code.~You~can~avoid~this~
11196   problem~by~using~'last-row'~without~value~(more~compilations~
11197   might~be~necessary).
11198 }
11199 \@@_msg_new:nn { Yet-in-env }
11200 {
11201   Nested~environments.\\
11202   Environments~of~nicematrix~can't~be~nested.~However~you~
11203   can~insert,~for~example,~a~\{tabular\}~in~a~\{NiceTabular\}~
11204   or~a~\{NiceTabular\}~in~a~\{tabular\}.~You~can~also~compose~
11205   an~environment~of~nicematrix~in~a~box~of~LaTeX~and~insert~
11206   that~box~in~another~environment~of~nicematrix.\\
11207   This~error~is~fatal.
11208 }
11209 \@@_msg_new:nn { Outside-math-mode }
11210 {
11211   Outside~math~mode.\\
11212   The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
11213   (and~not~in~ \token_to_str:N \vcenter ).\\
11214   This~error~is~fatal.
11215 }
11216 \@@_msg_new:nn { One-letter~allowed }
11217 {
11218   Bad~name.\\
11219   The~value~of~key~' \l_keys_key_str '~must~be~of~length~1~and~
11220   you~have~used~' \l_keys_value_tl '.\\
11221   It~will~be~ignored.
11222 }
11223 \@@_msg_new:nn { TabularNote~in~CodeAfter }
11224 {
11225   Environment~\{TabularNote\}~forbidden.\\
11226   You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
11227   but~*before*~the~ \token_to_str:N \CodeAfter . \\
11228   This~environment~\{TabularNote\}~will~be~ignored.
11229 }
11230 \@@_msg_new:nn { varwidth~not~loaded }
11231 {
11232   varwidth~not~loaded.\\
11233   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
11234   loaded.~You~should~load~'varwidth',~before~or~after~'nicematrix'. \\
11235   Your~column~will~behave~like~'p'.

```

```

11236 }
11237 \@@_msg_new:nn { varwidth~not~loaded~in~X }
11238 {
11239   varwidth~not~loaded.\\
11240   You~can't~use~the~key~'V'~in~your~column~'X'~
11241   because~'varwidth'~is~not~loaded.~You~should~load~'varwidth',~
11242   before~or~after~'nicematrix'.\\
11243   It~will~be~ignored. \\
11244 }
11245 \@@_msg_new:nnn { Unknown~key~for~a~rule }
11246 {
11247   Unknown~key.\\
11248   Your~key~' \l_keys_key_str '~is~unknown~for~a~rule.\\
11249   \c_@@_available_keys_str
11250 }
11251 {
11252   The~available~keys~are:~color,~multiplicity,~sep-color,~tikz~
11253   and~total-width~(meaningful~only~in~cunjunction~with~'tikz').
11254 }

```

If fact, there is also the key dotted but it won't be very useful since we provide `\hdottedline`, `\cdottedline` and the letter `:`.

```

11255 \@@_msg_new:nnn { Unknown~key~for~Block }
11256 {
11257   Unknown~key. \\
11258   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11259   \token_to_str:N \Block . \\
11260   It~will~be~ignored. \\
11261   \c_@@_available_keys_str
11262 }
11263 {
11264   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
11265   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~name,~
11266   opacity,~rounded~corners,~r,~respect~arraystretch,~rules/width,~t,~T,~tikz,~
11267   transparent~and~vlines.
11268 }
11269 \@@_msg_new:nnn { Unknown~key~for~Brace }
11270 {
11271   Unknown~key.\\
11272   The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
11273   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
11274   It~will~be~ignored. \\
11275   \c_@@_available_keys_str
11276 }
11277 {
11278   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
11279   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
11280   right~shorten)~and~yshift.
11281 }
11282 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
11283 {
11284   Unknown~key.\\
11285   The~key~' \l_keys_key_str '~is~unknown.\\
11286   It~will~be~ignored. \\
11287   \c_@@_available_keys_str
11288 }
11289 {
11290   The~available~keys~are~(in~alphabetic~order):~
11291   delimiters/color,~
11292   rules~(with~the~subkeys~'color'~and~'width'),~
11293   sub~matrix~(several~subkeys)~

```

```

11294     and~xdots~(several~subkeys).~
11295     The~latter~is~for~the~command~ \token_to_str:N \line .
11296 }

11297 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
11298 {
11299     Unknown~key.\\
11300     The~key~' \l_keys_key_str '~is~unknown.\\
11301     It~will~be~ignored. \\
11302     \c_@@_available_keys_str
11303 }
11304 {
11305     The~available~keys~are~(in~alphabetic~order):~
11306     create~cell~nodes,~
11307     delimiters/color~and~
11308     sub~matrix~(several~subkeys).
11309 }

11310 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
11311 {
11312     Unknown~key.\\
11313     The~key~' \l_keys_key_str '~is~unknown.\\
11314     That~key~will~be~ignored. \\
11315     \c_@@_available_keys_str
11316 }
11317 {
11318     The~available~keys~are~(in~alphabetic~order):~
11319     'delimiters/color',~
11320     'extra~height',~
11321     'hlines',~
11322     'hvlines',~
11323     'left~xshift',~
11324     'name',~
11325     'right~xshift',~
11326     'rules'~(with~the~subkeys~'color'~and~'width'),~
11327     'slim',~
11328     'vlines'~and~'xshift'~(which~sets~both~'left~xshift'~
11329     and~'right~xshift').\\
11330 }

11331 \@@_msg_new:nnn { Unknown~key~for~notes }
11332 {
11333     Unknown~key.\\
11334     The~key~' \l_keys_key_str '~is~unknown.\\
11335     That~key~will~be~ignored. \\
11336     \c_@@_available_keys_str
11337 }
11338 {
11339     The~available~keys~are~(in~alphabetic~order):~
11340     bottomrule,~
11341     code~after,~
11342     code~before(+),~
11343     detect~duplicates,~
11344     enumitem~keys,~
11345     enumitem~keys~para,~
11346     para,~
11347     label~in~list,~
11348     label~in~tabular~and~
11349     style.
11350 }

11351 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
11352 {
11353     Unknown~key.\\
11354     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11355     \token_to_str:N \RowStyle . \\

```

```

11356     That~key~will~be~ignored. \\
11357     \c_@@_available_keys_str
11358 }
11359 {
11360     The~available~keys~are~(in~alphabetic~order):~
11361     bold,~
11362     cell-space-top-limit(+),~
11363     cell-space-bottom-limit(+),~
11364     cell-space-limits(+),~
11365     color,~
11366     fill~(alias:~rowcolor),~
11367     nb-rows,~
11368     opacity~and~
11369     rounded-corners.
11370 }
11371 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
11372 {
11373     Unknown~key.\\
11374     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11375     \token_to_str:N \NiceMatrixOptions . \\
11376     That~key~will~be~ignored. \\
11377     \c_@@_available_keys_str
11378 }
11379 {
11380     The~available~keys~are~(in~alphabetic~order):~
11381     &~in~blocks,~
11382     allow-duplicate-names,~
11383     ampersand-in-blocks,~
11384     caption-above,~
11385     cell-space-bottom-limit(+),~
11386     cell-space-limits(+),~
11387     cell-space-top-limit(+),~
11388     code-for-first-col(+),~
11389     code-for-first-row(+),~
11390     code-for-last-col(+),~
11391     code-for-last-row(+),~
11392     corners,~
11393     custom-key,~
11394     create-extra-nodes,~
11395     create-medium-nodes,~
11396     create-large-nodes,~
11397     custom-line,~
11398     delimiters~(several~subkeys),~
11399     end-of-row,~
11400     first-col,~
11401     first-row,~
11402     h(v)lines,~
11403     h(v)lines-except-borders,~
11404     last-col,~
11405     last-row,~
11406     left-margin,~
11407     light-syntax,~
11408     light-syntax-expanded,~
11409     matrix/columns-type,~
11410     no-cell-nodes,~
11411     notes~(several~subkeys),~
11412     nullify-dots,~
11413     pgf-node-code,~
11414     renew-dots,~
11415     renew-matrix,~
11416     respect-arraystretch,~
11417     rounded-corners,~
11418     right-margin,~

```



```

11419 rules~(with~the~subkeys~'color'~and~'width'),~
11420 small,~
11421 sub-matrix~(several~subkeys),~
11422 vlines,~
11423 xdots~(several~subkeys).
11424 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

11425 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
11426 {
11427   Unknown~key.\\
11428   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11429   \{NiceArray\}. \\
11430   That~key~will~be~ignored. \\
11431   \c_@@_available_keys_str
11432 }
11433 {
11434   The~available~keys~are~(in~alphabetic~order):~
11435   &~in~blocks,~
11436   ampersand~in~blocks,~
11437   b,~
11438   baseline,~
11439   c,~
11440   cell-space-bottom-limit,~
11441   cell-space-limits,~
11442   cell-space-top-limit,~
11443   code-after,~
11444   code-for-first-col(+),~
11445   code-for-first-row(+),~
11446   code-for-last-col(+),~
11447   code-for-last-row(+),~
11448   columns-width,~
11449   corners,~
11450   create-blocks-in-col,~
11451   create-extra-nodes,~
11452   create-medium-nodes,~
11453   create-large-nodes,~
11454   draw-trees-in-col,~
11455   extra-left-margin,~
11456   extra-right-margin,~
11457   first-col,~
11458   first-row,~
11459   h(v)lines,~
11460   h(v)lines-except-borders,~
11461   last-col,~
11462   last-row,~
11463   left-margin,~
11464   light-syntax,~
11465   light-syntax-expanded,~
11466   name,~
11467   no-cell-nodes,~
11468   nullify-dots,~
11469   pgf-node-code,~
11470   renew-dots,~
11471   respect-arraystretch,~
11472   right-margin,~
11473   rounded-corners,~
11474   rules~(with~the~subkeys~'color'~and~'width'),~
11475   small,~
11476   t,~
11477   vlines,~
11478   xdots/color,~
11479   xdots/shorten-start(+),~

```

```

11480     xdots/shorten-end(+),~
11481     xdots/shorten(+)-and~
11482     xdots/line-style.
11483 }

```

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`).

```

11484 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
11485 {
11486   Unknown~key.\\
11487   The~key~' \l_keys_key_str '-is~unknown~for~the~
11488   \@@_full_name_env: . \\
11489   That~key~will~be~ignored. \\
11490   \c_@@_available_keys_str
11491 }
11492 {
11493   The~available~keys~are~(in~alphabetic~order):~
11494   &~in~blocks,~
11495   ampersand~in~blocks,~
11496   b,~
11497   baseline,~
11498   c,~
11499   cell-space-bottom-limit,~
11500   cell-space-limits,~
11501   cell-space-top-limit,~
11502   code-after,~
11503   code-for-first-col(+),~
11504   code-for-first-row(+),~
11505   code-for-last-col(+),~
11506   code-for-last-row(+),~
11507   columns-type,~
11508   columns-width,~
11509   corners,~
11510   create-blocks-in-col,~
11511   create-extra-nodes,~
11512   create-medium-nodes,~
11513   create-large-nodes,~
11514   draw-trees-in-col,~
11515   extra-left-margin,~
11516   extra-right-margin,~
11517   first-col,~
11518   first-row,~
11519   h(v)lines,~
11520   h(v)lines-except-borders,~
11521   l,~
11522   last-col,~
11523   last-row,~
11524   left-margin,~
11525   light-syntax,~
11526   light-syntax-expanded,~
11527   name,~
11528   no-cell-nodes,~
11529   nullify-dots,~
11530   pgf-node-code,~
11531   r,~
11532   renew-dots,~
11533   respect-arraystretch,~
11534   right-margin,~
11535   rounded-corners,~
11536   rules~(with~the~subkeys~'color'~and~'width'),~
11537   small,~
11538   t,~
11539   vlines,~

```

```

11540     xdots/color,~
11541     xdots/shorten-start(+),~
11542     xdots/shorten-end(+),~
11543     xdots/shorten(+)-and~
11544     xdots/line-style.
11545 }
11546 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
11547 {
11548     Unknown~key.\\
11549     The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11550     \{NiceTabular\}. \\
11551     That~key~will~be~ignored. \\
11552     \c_@@_available_keys_str
11553 }
11554 {
11555     The~available~keys~are~(in~alphabetic~order):~
11556     &~in~blocks,~
11557     ampersand~in~blocks,~
11558     b,~
11559     baseline,~
11560     c,~
11561     caption,~
11562     cell-space-bottom-limit,~
11563     cell-space-limits,~
11564     cell-space-top-limit,~
11565     code~after,~
11566     code~for~first~col(+),~
11567     code~for~first~row(+),~
11568     code~for~last~col(+),~
11569     code~for~last~row(+),~
11570     columns~width,~
11571     corners,~
11572     custom~line,~
11573     create~blocks~in~col,~
11574     create~extra~nodes,~
11575     create~medium~nodes,~
11576     create~large~nodes,~
11577     draw~trees~in~col,~
11578     extra~left~margin,~
11579     extra~right~margin,~
11580     first~col,~
11581     first~row,~
11582     h(v)lines,~
11583     h(v)lines~except~borders,~
11584     label,~
11585     last~col,~
11586     last~row,~
11587     left~margin,~
11588     light~syntax,~
11589     light~syntax~expanded,~
11590     name,~
11591     no~cell~nodes,~
11592     notes~(several~subkeys),~
11593     nullify~dots,~
11594     pgf~node~code,~
11595     renew~dots,~
11596     respect~arraystretch,~
11597     right~margin,~
11598     rounded~corners,~
11599     rules~(with~the~subkeys~'color'~and~'width'),~
11600     short~caption,~
11601     t,~
11602     tabularnote,~

```

```

11603     vlines,~
11604     xdots/color,~
11605     xdots/shorten-start(+),~
11606     xdots/shorten-end(+),~
11607     xdots/shorten(+)-and~
11608     xdots/line-style.
11609 }

11610 \@@_msg_new:nnn { Duplicate~name }
11611 {
11612   Duplicate~name.\\
11613   The~name~' \l_keys_value_tl '~is~already~used~and~you~shouldn't~use~
11614   the~same~environment~name~twice.~You~can~go~on,~but,~
11615   maybe,~you~will~have~incorrect~results~especially~
11616   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
11617   message~again,~use~the~key~'allow-duplicate-names'~in~
11618   ' \token_to_str:N \NiceMatrixOptions '~.\\
11619   \bool_if:NF \g_@@_messages_for_Overleaf_bool
11620   { For~a~list~of~the~names~already~used,~type~H~<return>. }
11621 }
11622 {
11623   The~names~already~defined~in~this~document~are:~
11624   \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
11625 }

11626 \@@_msg_new:nn { caption-above~in~env }
11627 {
11628   The~key~'caption-above'~must~be~used~in~\token_to_str:N \NiceMatrixOptions.\\
11629   That~key~will~be~ignored.
11630 }

11631 \@@_msg_new:nn { show-cell-names }
11632 {
11633   There~is~no~key~'show-cell-names'~in~nicematrix.\\
11634   You~should~use~the~command~\token_to_str:N \ShowCellNames\
11635   in~the~\token_to_str:N \CodeBefore\ or~the~\token_to_str:N
11636   \CodeAfter. \\
11637   That~key~will~be~ignored.
11638 }

11639 \@@_msg_new:nn { Option~auto~for~columns-width }
11640 {
11641   Erroneous~use.\\
11642   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
11643   That~key~will~be~ignored.
11644 }

11645 \@@_msg_new:nn { NiceTabularX~without~X }
11646 {
11647   NiceTabularX~without~X.\\
11648   You~should~not~use~\{NiceTabularX\}~without~X~columns.\\
11649   However,~you~can~go~on.
11650 }

11651 \@@_msg_new:nn { Preamble~forgotten }
11652 {
11653   Preamble~forgotten.\\
11654   You~have~probably~forgotten~the~preamble~of~your~
11655   \@@_full_name_env: . \\
11656   This~error~is~fatal.
11657 }

11658 \@@_msg_new:nn { Invalid~col~number }
11659 {
11660   Invalid~column~number.\\
11661   A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11662   specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.\\
11663   Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.

```

```

11664     }
11665 \@@_msg_new:nn { Invalid~row~number }
11666   {
11667     Invalid~row~number.\\
11668     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11669     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.\\
11670     Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
11671   }
11672 \@@_define_com:NNN p ( )
11673 \@@_define_com:NNN b [ ]
11674 \@@_define_com:NNN v | |
11675 \@@_define_com:NNN V \l \l
11676 \@@_define_com:NNN B \{ \}

```

Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	4
4	Parameters	9
5	The command <code>\tabularnote</code>	20
6	Command for creation of rectangle nodes	25
7	The options	26
8	Important code used by <code>{NiceArrayWithDelims}</code>	37
9	The <code>\CodeBefore</code>	53
10	The environment <code>{NiceArrayWithDelims}</code>	57
11	Construction of the preamble of the array	62
12	The redefinition of <code>\multicolumn</code>	79
13	The environment <code>{NiceMatrix}</code> and its variants	97
	13.1 Definition of <code>{pNiceMatrix}</code>	97
	13.2 The key <code>renew-matrix</code>	98
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	98
15	After the construction of the array	99
16	We draw the dotted lines	106
17	The actual instructions for drawing the dotted lines with <code>TikZ</code>	123
18	User commands available in the new environments	129
19	The command <code>\line</code> accessible in <code>\CodeAfter</code>	135
20	The command <code>\RowStyle</code>	137
21	Colors of cells, rows and columns	140
22	The vertical and horizontal rules	152
23	The empty corners	174
24	The environment <code>{NiceMatrixBlock}</code>	177
25	The extra nodes	178
26	The blocks	183
27	Automatic arrays	210
28	The redefinition of the command <code>\dotfill</code>	211
29	The command <code>\diagbox</code>	212

30	The keyword <code>\CodeAfter</code>	213
31	The delimiters in the preamble	214
32	The command <code>\SubMatrix</code>	215
33	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	224
34	The commands <code>HBrace</code> et <code>VBrace</code>	227
35	The command <code>TikzEveryCell</code>	230
36	The key <code>draw-trees-in-col</code>	232
37	The key <code>create-blocks-in-col</code>	233
38	The command <code>\ShowCellNames</code>	234
39	We process the options at package loading	236
40	About the package underscore	237
41	Compatibility with <code>threeparttable</code>	238
42	Error messages of the package	238