

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

October 21, 2025

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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 7.3 of `nicematrix`, at the date of 2025/09/30.

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 }

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

```

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

```

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

```

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

```

36 \cs_new_protected:Npn \@@_error_or_warning:n
37 {
38   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
39     { \@@_warning:n }
40     { \@@_error:n }
41 }

```

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

```

42 \bool_new:N \g_@@_messages_for_Overleaf_bool
43 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
44 {
45   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
46   || \str_if_eq_p:ee \c_sys_jobname_str { output } % for Overleaf
47 }

48 \@@_msg_new:nn { mdwtab-loaded }
49 {
50   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
51   This~error~is~fatal.
52 }

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

```

2 Collecting options

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

Example :

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

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

Therefore, by writing : `\def\G{\@@_collect_options:n{\F}}`,

the command `\G` takes in an arbitrary number of optional arguments between square brackets.

Be careful: that command is *not* “fully expandable” (because of `\peek_meaning:NTF`).

```

55 \cs_new_protected:Npn \@@_collect_options:n #1
56 {
57   \peek_meaning:NTF [
58     { \@@_collect_options:nw { #1 } }
59     { #1 { } }
60   }

```

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

```

61 \NewDocumentCommand \@@_collect_options:nw { m r[] }
62 { \@@_collect_options:nn { #1 } { #2 } }
63
64 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
65 {
66   \peek_meaning:NTF [
67     { \@@_collect_options:nnw { #1 } { #2 } }
68     { #1 { #2 } }
69   }
70
71 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
72 { \@@_collect_options:nn { #1 } { #2 , #3 } }

```

3 Technical definitions

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

```

73 \tl_const:Nn \c_@@_c_tl { c }
74 \tl_const:Nn \c_@@_l_tl { l }
75 \tl_const:Nn \c_@@_r_tl { r }
76 \tl_const:Nn \c_@@_all_tl { all }
77 \tl_const:Nn \c_@@_dot_tl { . }
78 \str_const:Nn \c_@@_r_str { r }
79 \str_const:Nn \c_@@_c_str { c }
80 \str_const:Nn \c_@@_l_str { l }

```

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).

```

81 \tl_new:N \l_@@_argspec_tl

```

```

82 \cs_generate_variant:Nn \seq_set_split:Nnn { N o }
83 \cs_generate_variant:Nn \str_set:Nn { N o }
84 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
85 \prg_generate_conditional_variant:Nnn \clist_if_in:Nn { N e } { T , F, TF }
86 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { e } { T }
87 \prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
88 \cs_generate_variant:Nn \dim_min:nn { v }
89 \cs_generate_variant:Nn \dim_max:nn { v }

90 \hook_gput_code:nnn { begindocument } { . }
91 {
92   \IfPackageLoadedTF { tikz }
93   {

```

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

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

```

94   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
95   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
96 }
97 {
98   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
99   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
100 }
101 }

```

We test whether the current class is `revtex4-1` (deprecated) or `revtex4-2` because these classes redefines `\array` (of `array`) in a way incompatible with our programming. At the date April 2025, the current version `revtex4-2` is 4.2f (compatible with `booktabs`).

```

102 \IfClassLoadedTF { revtex4-1 }
103 { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
104 {
105   \IfClassLoadedTF { revtex4-2 }
106   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
107   {

```

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

```

108   \cs_if_exist:NT \rvtx@ifformat@geq
109   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
110   { \bool_const:Nn \c_@@_revtex_bool { \c_false_bool } }
111 }
112 }

```

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.

```

113 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
114 {
115   \iow_now:Nn \@mainaux
116   {
117     \ExplSyntaxOn
118     \cs_if_free:NT \pgfsyspdfmark
119     { \cs_set_eq:NN \pgfsyspdfmark \gobblethree }
120     \ExplSyntaxOff
121   }
122   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
123 }

```

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

```

124 \ProvideDocumentCommand \iddots { }
125 {
126   \mathinner
127   {
128     \mkern 1 mu
129     \box_move_up:nn { 1 pt } { \hbox { . } }
130     \mkern 2 mu
131     \box_move_up:nn { 4 pt } { \hbox { . } }
132     \mkern 2 mu
133     \box_move_up:nn { 7 pt }
134     { \vbox:n { \kern 7 pt \hbox { . } } }
135     \mkern 1 mu
136   }
137 }

```

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.

```

138 \hook_gput_code:nnn { begindocument } { . }
139 {
140   \IfPackageLoadedT { booktabs }
141   { \iow_now:Nn \@mainaux { \nicematrix@redefine@check@rerun } }
142 }
143 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
144 {
145   \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`).

```

146   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
147   {
148     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } { 1 } { 3 } }
149     { \@_old_pgfutil@check@rerun { ##1 } { ##2 } }
150   }
151 }

```

We have to know whether `colortbl` is loaded in particular for the redefinition of `\everycr`.

```

152 \hook_gput_code:nnn { begindocument } { . }
153 {
154   \cs_set_protected:Npe @@_everycr:
155   {
156     \IfPackageLoadedTF { colortbl } { \CT@everycr } { \everycr }
157     { \noalign { @@_in_everycr: } }
158   }
159   \IfPackageLoadedTF { colortbl }
160   {
161     \cs_set_eq:NN @@_old_cellcolor: \cellcolor
162     \cs_set_eq:NN @@_old_rowcolor: \rowcolor
163     \cs_new_protected:Npn @@_revert_colortbl:
164     {
165       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
166       {
167         \cs_set_eq:NN \cellcolor @@_old_cellcolor:
168         \cs_set_eq:NN \rowcolor @@_old_rowcolor:

```

```

169         }
170     }

```

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).

```

171     \cs_new_protected:Npn \@@_replace_columncolor:
172     {
173         \tl_replace_all:Nnn \g_@@_array_preamble_tl
174         { \columncolor }
175         { \@@_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.

```

176     }
177 }
178 {
179     \cs_new_protected:Npn \@@_revert_colortbl: { }
180     \cs_new_protected:Npn \@@_replace_columncolor:
181     { \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.

```

182     \def \CT@arc@ { }
183     \def \arrayrulecolor #1 # { \CT@arc@ { #1 } }
184     \def \CT@arc@ #1 #2
185     {
186         \dim_compare:nNnT { \baselineskip } = { \c_zero_dim } { \noalign }
187         { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
188     }

```

Idem for `\CT@drs@`.

```

189     \def \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
190     \def \CT@drs@ #1 #2
191     {
192         \dim_compare:nNnT { \baselineskip } = { \c_zero_dim } { \noalign }
193         { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
194     }
195     \def \hline
196     {
197         \noalign { \ifnum 0 = ` } \fi
198         \cs_set_eq:NN \hskip \vskip
199         \cs_set_eq:NN \vrule \hrule
200         \cs_set_eq:NN \@width \@height
201         { \CT@arc@ \vline }
202         \futurelet \reserved@a
203         \@xhline
204     }
205 }
206 }

```

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.

```

207 \cs_set_nopar:Npn \@@_standard_cline: #1 { \@@_standard_cline:w #1 \q_stop }
208 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
209 {
210     \int_if_zero:nT { \l_@@_first_col_int } { \omit & }
211     \int_compare:nNnT { #1 } > { \c_one_int }
212     { \multispan { \int_eval:n { #1 - 1 } } & }
213     \multispan { \int_eval:n { #2 - #1 + 1 } }
214     {
215         \CT@arc@
216         \leaders \hrule \@height \arrayrulewidth \hfill

```

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

```
217     \skip_horizontal:N \c_zero_dim
218 }
```

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`.

```
219     \everycr { }
220     \cr
221     \noalign { \skip_vertical:n { - \arrayrulewidth } }
222 }
```

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.

```
223 \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`.

```
224 { \@@_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*.

```
225 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
226 \cs_generate_variant:Nn \@@_cline_i:nn { e }
227 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
228 {
229     \tl_if_empty:nTF { #3 }
230     { \@@_cline_iii:w #1|#2-#2 \q_stop }
231     { \@@_cline_ii:w #1|#2-#3 \q_stop }
232 }
233 \cs_set:Npn \@@_cline_ii:w #1|#2-#3- \q_stop
234 { \@@_cline_iii:w #1|#2-#3 \q_stop }
235 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
236 {
```

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).

```
237     \int_compare:nNnT { #1 } < { #2 }
238     { \multispan { \int_eval:n { #2 - #1 } } & }
239     \multispan { \int_eval:n { #3 - #2 + 1 } }
240     {
241         \CT@arc@
242         \leaders \hrule \@height \arrayrulewidth \hfill
243         \skip_horizontal:N \c_zero_dim
244     }
```

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

```
245     \peek_meaning_remove_ignore_spaces:NNTF \cline
246     { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
247     { \everycr { } \cr }
248 }
```

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

```
249 \cs_set_eq:NN \@@_math_toggle: \c_math_toggle_token
```

¹See question 99041 on TeX StackExchange.

```

250 \cs_new_protected:Npn \@@_set_CTarc:n #1
251 {
252   \tl_if_blank:nF { #1 }
253   {
254     \tl_if_head_eq_meaning:nNTF { #1 } [
255       { \def \CT@arc@ { \color #1 } }
256       { \def \CT@arc@ { \color { #1 } } }
257     ]
258   }
259 \cs_generate_variant:Nn \@@_set_CTarc:n { o }

```

```

260 \cs_new_protected:Npn \@@_set_CTdrsc:n #1
261 {
262   \tl_if_head_eq_meaning:nNTF { #1 } [
263     { \def \CT@drsc@ { \color #1 } }
264     { \def \CT@drsc@ { \color { #1 } } }
265   ]

```

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

```

266 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
267 {
268   \tl_if_head_eq_meaning:nNTF { #2 } [
269     { #1 #2 }
270     { #1 { #2 } }
271   ]
272 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```

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

```

273 \cs_new_protected:Npn \@@_color:n #1
274 { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }
275 \cs_generate_variant:Nn \@@_color:n { o }

```

```

276 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
277 {
278   \tl_set_rescan:Nno
279     #1
280     {
281       \char_set_catcode_other:N >
282       \char_set_catcode_other:N <
283     }
284   #1
285 }

```

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

```

286 \dim_new:N \l_@@_tmpc_dim
287 \dim_new:N \l_@@_tmpd_dim
288 \tl_new:N \l_@@_tmpc_tl
289 \tl_new:N \l_@@_tmpd_tl
290 \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.

```
291 \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).

```
292 \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.

```
293 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
294 { \int_use:N \g_@@_env_int }
```

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

```
295 \cs_new_protected:Npn \@@_qpoint:n #1
296 { \pgfpointanchor { \@@_env: - #1 } { center } }
```

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

```
297 \bool_new:N \l_@@_tabular_bool
```

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

```
298 \bool_new:N \g_@@_delims_bool
299 \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.

```
300 \bool_new:N \l_@@_preamble_bool
301 \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.

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

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

```
303 \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.

```
304 \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).

```
305 \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.).

```
306 \dim_new:N \l_@@_col_width_dim
307 \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.

```
308 \int_new:N \g_@@_row_total_int
309 \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).

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

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

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

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

```
312 \tl_new:N \l_@@_hpos_cell_tl
313 \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`.

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

Idem for the mono-row blocks.

```
315 \dim_new:N \g_@@_blocks_ht_dim
316 \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}`).

```
317 \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`.

```
318 \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.

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

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

```
320 \bool_new:N \l_@@_notes_detect_duplicates_bool
321 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

```
322 \bool_new:N \l_@@_initial_open_bool
323 \bool_new:N \l_@@_final_open_bool
324 \bool_new:N \l_@@_Vbrace_bool
```

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

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

The following dimension will be used for the total width of composite rules (*total* means that the spaces on both sides are included).

```
326 \dim_new:N \l_@@_rule_width_dim
```

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

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

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

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

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

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

In a cell, it will be possible to know whether we are in a cell of a column of type `X` thanks to that flag (the `X` columns of `nicematrix` are inspired by those of `tabularx`). You will use that flag for the blocks.

```
330 \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`).

```
331 \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`.

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

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

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

```
334 \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 }`).

```
335 \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.

```
336 \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.

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

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

```
339 \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}`).

```
340 \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`).

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

For `\multicolumn`.

```
342 \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`.

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

```
344 \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 `:`.

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

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

```
347 \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`).

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

```
349 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
350 {
```

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

```
352     \@@_fatal:n { Outside~math~mode }
```

```
353     \fi:
```

```
354 }
```

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 analyse of the preamble of the array.

```
355 \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”.

```
356 \colorlet { nicematrix-last-col } { . }
```

```
357 \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`).

```
358 \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*.

```
359 \str_new:N \g_@@_com_or_env_str
```

```
360 \str_gset:Nn \g_@@_com_or_env_str { environment }
```

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

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

```
362 \cs_new:Npn \@@_full_name_env:
```

```
363 {
```

```
364     \str_if_eq:eeTF { \g_@@_com_or_env_str } { command }
```

```
365     { command \space \c_backslash_str \g_@@_name_env_str }
```

```
366     { environment \space \{ \g_@@_name_env_str \} }
```

```
367 }
```

```
368 \tl_new:N \g_@@_cell_after_hook_tl % 2025/03/22
```

For the key `code` of the command `\SubMatrix` (itself in the main `\CodeAfter`), we will use the following token list.

```
369 \tl_new:N \l_@@_code_tl
```

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

```
370 \tl_new:N \l_@@_pgf_node_code_tl
```

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

```
371 \tl_new:N \g_@@_pre_code_before_tl
372 \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`.

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

```
373 \tl_new:N \g_@@_pre_code_after_tl
374 \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.

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

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

```
376 \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.

```
377 \int_new:N \l_@@_old_iRow_int
378 \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).

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

The following token list corresponds to the key `rules/color` available in the environments.

```
380 \tl_new:N \l_@@_rules_color_tl
```

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

```
381 \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.

```
382 \bool_new:N \l_@@_X_columns_aux_bool
383 \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.

```
384 \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).

```
385 \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).

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

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

```
388 \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).

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

The following dimensions will be used when drawing the dotted lines.

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

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

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

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

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

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

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

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

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

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

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

```
400 \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”.

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

```
402 \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.

```
403 \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!).

```
404 \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.

```
405 \seq_new:N \g_@@_future_pos_of_blocks_seq
```

The, after the execution of the `\CodeBefore`, the sequence `\g_@@_pos_of_blocs_seq` will be erased and replaced by the value of `\g_@@_future_pos_of_blocks_seq`.

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}`.

```
406 \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.

```
407 \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`).

```
408 \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`).

```
409 \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.

```
410 \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).

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

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

By default, the diagonal lines will be parallelized². There are two types of diagonals lines: the `\Ddots` diagonals and the `\Iddots` diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current `{NiceArray}` environment.

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

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

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

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

```

415 \dim_new:N \g_@@_delta_x_one_dim
416 \dim_new:N \g_@@_delta_y_one_dim
417 \dim_new:N \g_@@_delta_x_two_dim
418 \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`).

```

419 \int_new:N \l_@@_row_min_int
420 \int_new:N \l_@@_row_max_int
421 \int_new:N \l_@@_col_min_int
422 \int_new:N \l_@@_col_max_int

423 \int_new:N \l_@@_initial_i_int
424 \int_new:N \l_@@_initial_j_int
425 \int_new:N \l_@@_final_i_int
426 \int_new:N \l_@@_final_j_int

```

The following counters will be used when drawing the rules.

```

427 \int_new:N \l_@@_start_int
428 \int_set_eq:NN \l_@@_start_int \c_one_int
429 \int_new:N \l_@@_end_int
430 \int_new:N \l_@@_local_start_int
431 \int_new:N \l_@@_local_end_int

```

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

```

432 \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).

```

433 \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`.

```

434 \tl_new:N \l_@@_fill_tl
435 \tl_new:N \l_@@_opacity_tl
436 \tl_new:N \l_@@_draw_tl
437 \seq_new:N \l_@@_tikz_seq
438 \clist_new:N \l_@@_borders_clist
439 \dim_new:N \l_@@_rounded_corners_dim

```

The last parameter has no direct link with the [empty] corners of the array (which are computed and taken into account by `nicematrix` when the key `corners` is used).

The following dimension corresponds to the key `rounded-corners` available in an individual environment `{NiceTabular}`. When that key is used, a clipping is applied in the `\CodeBefore` of the environment in order to have rounded corners for the potential colored panels.

```

440 \dim_new:N \l_@@_tab_rounded_corners_dim

```

The following token list correspond to the key `color` of the command `\Block` and also the key `color` of the command `\RowStyle`.

```

441 \tl_new:N \l_@@_color_tl

```


In the key `tikz` of a command `\Block` or in the argument of a command `\TikzEveryCell`, the final user puts a list of `tikz` keys. But, you have added another key, named `offset` (which means that an offset will be used for the frame of the block or the cell). The following parameter corresponds to that key.

```
442 \dim_new:N \l_@@_offset_dim
```

Here is the dimension for the width of the rule when a block (created by `\Block`) is stroked or when the key `hvlines` is used.

```
443 \dim_new:N \l_@@_line_width_dim
```

The parameters of the horizontal position of the label of a block. If the user uses the key `c` or `C`, the value is `c`. If the user uses the key `l` or `L`, the value is `l`. If the user uses the key `r` or `R`, the value is `r`. If the user has used a capital letter, the boolean `\l_@@_hpos_of_block_cap_bool` will be raised (in the second pass of the analyze of the keys of the command `\Block`).

```
444 \str_new:N \l_@@_hpos_block_str
445 \str_set:Nn \l_@@_hpos_block_str { c }
446 \bool_new:N \l_@@_hpos_of_block_cap_bool
447 \bool_new:N \l_@@_p_block_bool
```

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

```
448 \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).

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

Used when the key `draw-first` is used for `\Ddots` or `\Iddots`.

```
450 \bool_new:N \l_@@_draw_first_bool
```

The following flag corresponds to the keys `vlines` and `hlines` of the command `\Block` (the key `hvlines` is the conjunction of both).

```
451 \bool_new:N \l_@@_vlines_block_bool
452 \bool_new:N \l_@@_hlines_block_bool
```

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

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

454 \dim_new:N \l_@@_submatrix_extra_height_dim
455 \dim_new:N \l_@@_submatrix_left_xshift_dim
456 \dim_new:N \l_@@_submatrix_right_xshift_dim
457 \clist_new:N \l_@@_hlines_clist
458 \clist_new:N \l_@@_vlines_clist
459 \clist_new:N \l_@@_submatrix_hlines_clist
460 \clist_new:N \l_@@_submatrix_vlines_clist
```

The following key is set when the keys `hvlines` and `hvlines-except-borders` are used. It’s used only to change slightly the clipping path set by the key `rounded-corners` (for a `{tabular}`).

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

The following flag will be used by (for instance) `\@@_vline_ii:`. When `\l_@@_dotted_bool` is `true`, a dotted line (with our system) will be drawn.

```
462 \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`).

```
463 \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.

```
464 \int_new:N \l_@@_first_row_int
465 \int_set_eq:NN \l_@@_first_row_int \c_one_int
```

- **First column**

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

```
466 \int_new:N \l_@@_first_col_int
467 \int_set_eq:NN \l_@@_first_col_int \c_one_int
```

- **Last row**

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

```
468 \int_new:N \l_@@_last_row_int
469 \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

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

Idem for `\l_@@_last_col_without_value_bool`

```
471 \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.

```
472 \int_new:N \l_@@_last_col_int
473 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

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.

```

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

```

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

```

474 \bool_new:N \g_@@_last_col_found_bool

```

This boolean is set to `false` at the end of `\@@_pre_array_ii:`.

In the last column, we will raise the following flag (it will be used by `\OnlyMainNiceMatrix`).

```

475 \bool_new:N \l_@@_in_last_col_bool

```

Some utilities

```

476 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
477 {

```

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

```

478 \def \l_tmpa_tl { #1 }
479 \def \l_tmpb_tl { #2 }
480 }

```

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`.

```

481 \cs_new_protected:Npn \@@_expand_clist_hvlines:NN #1 #2
482 {
483 \clist_if_in:NnF #1 { all }
484 {
485 \clist_clear:N \l_tmpa_clist
486 \clist_map_inline:Nn #1
487 {
488 \tl_if_head_eq_meaning:nNTF { ##1 } -
489 {

```

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.

```

490 \int_if_zero:nF { #2 }
491 {
492 \clist_put_right:Ne \l_tmpa_clist
493 { \int_eval:n { #2 + (##1) + 1 } }
494 }
495 }
496 {

```

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

```

497 \tl_if_in:nnTF { ##1 } { - }
498 { \@@_cut_on_hyphen:w ##1 \q_stop }
499 {

```

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

```

500 \def \l_tmpa_tl { ##1 }
501 \def \l_tmpb_tl { ##1 }
502 }
503 \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
504 { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
505 }
506 }
507 \tl_set_eq:NN #1 \l_tmpa_clist
508 }
509 }

```

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.

```

510 \hook_gput_code:nnn { begindocument } { . }
511 {
512   \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
513   \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
514 }

```

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.

```

515 \newcounter { tabularnote }

```

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

```

516 \int_new:N \g_@@_tabularnote_int
517 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }

518 \seq_new:N \g_@@_notes_seq
519 \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.

```

520 \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.

```

521 \seq_new:N \l_@@_notes_labels_seq
522 \newcounter { nicematrix_draft }
523 \cs_new_protected:Npn \@@_notes_format:n #1
524 {
525   \setcounter { nicematrix_draft } { #1 }
526   \@@_notes_style:n { nicematrix_draft }
527 }

```

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

```

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

```

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

```

529 \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`.

```

530 \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`.

```

531 \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`).

```

532 \hook_gput_code:nnn { begindocument } { . }
533 {
534   \IfPackageLoadedTF { enumitem }
535   {

```

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.

```

536     \newlist { tabularnotes } { enumerate } { 1 }
537     \setlist [ tabularnotes ]
538     {
539       topsep = \c_zero_dim ,
540       noitemsep ,
541       leftmargin = * ,
542       align = left ,
543       labelsep = \c_zero_dim ,
544       label =
545         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
546     }

```

```

547 \newlist { tabularnotes* } { enumerate* } { 1 }
548 \setlist [ tabularnotes* ]
549 {
550     afterlabel = \nobreak ,
551     itemjoin = \quad ,
552     label =
553     \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
554 }

```

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`.

```

555 \NewDocumentCommand \tabularnote { o m }
556 {
557     \bool_lazy_or:nnT { \cs_if_exist_p:N \@capytype } { \l_@@_in_env_bool }
558     {
559         \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } { \l_@@_in_env_bool }
560         { \@@_error:n { tabularnote~forbidden } }
561         {
562             \bool_if:NTF \l_@@_in_caption_bool
563             \@@_tabularnote_caption:nn
564             \@@_tabularnote:nn
565             { #1 } { #2 }
566         }
567     }
568 }
569 {
570     \NewDocumentCommand \tabularnote { o m }
571     { \@@_err_enumitem_not_loaded: }
572 }
573 }
574 }
575 \cs_new_protected:Npn \@@_err_enumitem_not_loaded:
576 {
577     \@@_error_or_warning:n { enumitem~not~loaded }
578     \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
579 }
580 \cs_new_protected:Npn \@@_test_first_novaluen { #1 #2 #3
581     { \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`.

```

582 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
583 {

```

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).

```

584 \int_zero:N \l_tmpa_int
585 \bool_if:NT \l_@@_notes_detect_duplicates_bool
586 {

```

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`.

```

587     \int_zero:N \l_tmpb_int
588     \seq_map_indexed_inline:Nn \g_@@_notes_seq
589     {
590         \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
591         \tl_if_eq:nnT { { #1 } { #2 } } { { ##2 }
592             {
593                 \tl_if_novalue:nTF { #1 }
594                 { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
595                 { \int_set:Nn \l_tmpa_int { ##1 } }
596                 \seq_map_break:
597             }
598         }
599         \int_if_zero:nF { \l_tmpa_int }
600         { \int_add:Nn \l_tmpa_int { \g_@@_notes_caption_int } }
601     }
602     \int_if_zero:nT { \l_tmpa_int }
603     {
604         \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
605         \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
606     }
607     \seq_put_right:Ne \l_@@_notes_labels_seq
608     {
609         \tl_if_novalue:nTF { #1 }
610         {
611             \@@_notes_format:n
612             {
613                 \int_eval:n
614                 {
615                     \int_if_zero:nTF { \l_tmpa_int }
616                     { \c@tabularnote }
617                     { \l_tmpa_int }
618                 }
619             }
620         }
621         { #1 }
622     }
623     \peek_meaning:NF \tabularnote
624     {

```

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`.

```

625         \hbox_set:Nn \l_tmpa_box
626         {

```

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

```

627         \@@_notes_label_in_tabular:n
628         {
629             \seq_use:Nnnn
630             \l_@@_notes_labels_seq { , } { , } { , }
631         }
632     }

```

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

```

633     \int_gdecr:N \c@tabularnote
634     \int_set_eq:NN \l_tmpa_int \c@tabularnote

```

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

```

635      \int_gincr:N \g_@@_tabularnote_int
636      \refstepcounter { tabularnote }
637      \int_compare:nNnT { \l_tmpa_int } = { \c@tabularnote }
638      { \int_gincr:N \c@tabularnote }
639      \seq_clear:N \l_@@_notes_labels_seq
640      \bool_lazy_or:nnTF
641      { \str_if_eq_p:ee \l_@@_hpos_cell_tl { c } }
642      { \str_if_eq_p:ee \l_@@_hpos_cell_tl { r } }
643      {
644      \hbox_overlap_right:n { \box_use:N \l_tmpa_box }

```

If the command `\tabularnote` is used exactly at the end of the cell, the `\unskip` (inserted by `array`?) will delete the skip we insert now and the label of the footnote will be composed in an overlapping position (by design).

```

645      \skip_horizontal:n { \box_wd:N \l_tmpa_box }
646      }
647      { \box_use:N \l_tmpa_box }
648  }
649 }

```

Now the version when the command is used in the key `caption`. The main difficulty is that the argument of the command `\caption` is composed several times. In order to know the number of commands `\tabularnote` in the caption, we will consider that there should not be the same tabular note twice in the caption (in the main tabular, it's possible). Once we have found a tabular note which has yet been encountered, we consider that you are in a new composition of the argument of `\caption`.

```

650 \cs_new_protected:Npn \g_@@_tabularnote_caption:nn #1 #2
651 {
652   \bool_if:NTF \g_@@_caption_finished_bool
653   {
654     \int_compare:nNnT { \c@tabularnote } = { \g_@@_notes_caption_int }
655     { \int_gzero:N \c@tabularnote }

```

Now, we try to detect duplicate notes in the caption. Be careful! We must put `\tl_if_in:NnF` and not `\tl_if_in:NnT`!

```

656   \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
657   { \@@_error:n { Identical-notes-in-caption } }
658 }
659 {

```

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

```

660   \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
661   {

```

Now, we know that are in the second composition of the caption since we are reading a tabular note which has yet been read. Now, the value of `\g_@@_notes_caption_int` won't change anymore: it's the number of uses *without optional argument* of the command `\tabularnote` in the caption.

```

662     \bool_gset_true:N \g_@@_caption_finished_bool
663     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
664     \int_gzero:N \c@tabularnote
665   }
666   { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
667 }

```

Now, we will compose the label of the footnote (in the caption). Even if we are not in the first composition, we have to compose that label!

```

668   \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
669   \seq_put_right:Ne \l_@@_notes_labels_seq
670   {
671     \tl_if_novalue:nTF { #1 }

```



```

672         { \@@_notes_format:n { \int_use:N \c@tabularnote } }
673         { #1 }
674     }
675     \peek_meaning:NF \tabularnote
676     {
677         \@@_notes_label_in_tabular:n
678         { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
679         \seq_clear:N \l_@@_notes_labels_seq
680     }
681 }

682 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
683 { \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.

```

684 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
685 {
686     \begin { pgfscope }
687     \pgfset
688     {
689         inner~sep = \c_zero_dim ,
690         minimum~size = \c_zero_dim
691     }
692     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
693     \pgfnode
694     { rectangle }
695     { center }
696     {
697         \vbox_to_ht:nn
698         { \dim_abs:n { #5 - #3 } }
699         {
700             \vfill
701             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { } { }
702         }
703     }
704     { #1 }
705     { }
706     \end { pgfscope }
707 }

```

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.

```

708 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
709 {
710     \begin { pgfscope }
711     \pgfset
712     {
713         inner~sep = \c_zero_dim ,
714         minimum~size = \c_zero_dim
715     }
716     \pgftransformshift { \pgfpoint scale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
717     \pgfpointdiff { #3 } { #2 }
718     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
719     \pgfnode
720     { rectangle }

```

```

721     { center }
722     {
723         \vbox_to_ht:nn
724         { \dim_abs:n \l_tmpb_dim }
725         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
726     }
727     { #1 }
728     { }
729 \end { pgfscope }
730 }

```

7 The options

The following parameter corresponds to the keys `caption`, `short-caption` and `label` of the environment `{NiceTabular}`.

```

731 \tl_new:N \l_@@_caption_tl
732 \tl_new:N \l_@@_short_caption_tl
733 \tl_new:N \l_@@_label_tl

```

The following parameter corresponds to the key `caption-above` of `\NiceMatrixOptions`. When this parameter is `true`, the captions of the environments `{NiceTabular}`, specified with the key `caption` are put above the tabular (and below elsewhere).

```

734 \bool_new:N \l_@@_caption_above_bool

```

By default, the behaviour of `\cline` is changed in the environments of `nicematrix`: a `\cline` spreads the array by an amount equal to `\arrayrulewidth`. It's possible to disable this feature with the key `\l_@@_standard_line_bool`.

```

735 \bool_new:N \l_@@_standard_cline_bool

```

The following dimensions correspond to the options `cell-space-top-limit` and `co` (these parameters are inspired by the package `cellspace`).

```

736 \dim_new:N \l_@@_cell_space_top_limit_dim
737 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

The following parameter corresponds to the key `xdots/horizontal_labels`.

```

738 \bool_new:N \l_@@_xdots_h_labels_bool

```

The following dimension is the distance between two dots for the dotted lines (when `line-style` is equal to `standard`, which is the initial value). The initial value is 0.45 em but it will be changed if the option `small` is used.

```

739 \dim_new:N \l_@@_xdots_inter_dim
740 \hook_gput_code:nnn { begindocument } { . }
741 { \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).

```

742 \dim_new:N \l_@@_xdots_shorten_start_dim
743 \dim_new:N \l_@@_xdots_shorten_end_dim
744 \hook_gput_code:nnn { begindocument } { . }
745 {
746     \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
747     \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
748 }

```

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.

```
749 \dim_new:N \l_@@_xdots_radius_dim
750 \hook_gput_code:nnn { begindocument } { . }
751 { \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.

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

```
755 \bool_new:N \l_@@_light_syntax_bool
756 \bool_new:N \l_@@_light_syntax_expanded_bool
```

The string `\l_@@_baseline_tl` may contain one of the three values `t`, `c` or `b` as in the option of the environment `{array}`. However, it may also contain **an integer** (which represents the number of the row to which align the array).

```
757 \tl_new:N \l_@@_baseline_tl
758 \tl_set:Nn \l_@@_baseline_tl { c }
```

The following parameter corresponds to the key `ampersand-in-blocks`

```
759 \bool_new:N \l_@@_amp_in_blocks_bool
```

The flag `\l_@@_exterior_arraycolsep_bool` corresponds to the option `exterior-arraycolsep`. If this option is set, a space equal to `\arraycolsep` will be put on both sides of an environment `{NiceArray}` (as it is done in `{array}` of `array`).

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

The flag `\l_@@_parallelize_diags_bool` controls whether the diagonals are parallelized. The initial value is `true`.

```
761 \bool_new:N \l_@@_parallelize_diags_bool
762 \bool_set_true:N \l_@@_parallelize_diags_bool
```

The following parameter correspond to the key `corners`. The elements of that `clist` must be within `NW`, `SW`, `NE` and `SE`.

```
763 \clist_new:N \l_@@_corners_clist
```

The flag `\l_@@_nullify_dots_bool` corresponds to the option `nullify-dots`. When the flag is down, the instructions like `\vdots` are inserted within a `\hphantom` (and so the constructed matrix has exactly the same size as a matrix constructed with the classical `{matrix}` and `\ldots`, `\vdots`, etc.).

```
764 \bool_new:N \l_@@_nullify_dots_bool
```

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

```
765 \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).

```
766 \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!

```
767 \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.

```
768 \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”.

```
769 \bool_new:N \l_@@_medium_nodes_bool
```

```
770 \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).

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

The dimension `\l_@@_left_margin_dim` correspond to the option `left-margin`. Idem for the right margin. These parameters are involved in the creation of the “medium nodes” but also in the placement of the delimiters and the drawing of the horizontal dotted lines (`\hdottedline`).

```
772 \dim_new:N \l_@@_left_margin_dim
```

```
773 \dim_new:N \l_@@_right_margin_dim
```

The dimensions `\l_@@_extra_left_margin_dim` and `\l_@@_extra_right_margin_dim` correspond to the options `extra-left-margin` and `extra-right-margin`.

```
774 \dim_new:N \l_@@_extra_left_margin_dim
```

```
775 \dim_new:N \l_@@_extra_right_margin_dim
```

The token list `\l_@@_end_of_row_tl` corresponds to the option `end-of-row`. It specifies the symbol used to mark the ends of rows when the light syntax is used.

```
776 \tl_new:N \l_@@_end_of_row_tl
```

```
777 \tl_set:Nn \l_@@_end_of_row_tl { ; }
```

The following parameter is for the color the dotted lines drawn by `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` but *not* the dotted lines drawn by `\hdottedline` and “:”.

```
778 \tl_new:N \l_@@_xdots_color_tl
```

The following token list corresponds to the key `delimiters/color`.

```
779 \tl_new:N \l_@@_delimiters_color_tl
```

Sometimes, we want to have several arrays vertically juxtaposed in order to have an alignment of the columns of these arrays. To achieve this goal, one may wish to use the same width for all the columns (for example with the option `columns-width` or the option `auto-columns-width` of the environment `{NiceMatrixBlock}`). However, even if we use the same type of delimiters, the width of the delimiters may be different from an array to another because the width of the delimiter is 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. The following boolean corresponds to this option.

```
780 \bool_new:N \l_@@_delimiters_max_width_bool
```

```

781 \keys_define:nn { nicematrix / xdots }
782 {
783   Vbrace .bool_set:N = \l_@@_Vbrace_bool ,
784   shorten-start .code:n =
785     \hook_gput_code:nnn { begindocument } { . }
786     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
787   shorten-end .code:n =
788     \hook_gput_code:nnn { begindocument } { . }
789     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
790   shorten-start .value_required:n = true ,
791   shorten-end .value_required:n = true ,
792   shorten .code:n =
793     \hook_gput_code:nnn { begindocument } { . }
794     {
795       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
796       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
797     } ,
798   shorten .value_required:n = true ,
799   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
800   horizontal-labels .default:n = true ,
801   horizontal-label .bool_set:N = \l_@@_xdots_h_labels_bool ,
802   horizontal-label .default:n = true ,
803   line-style .code:n =
804     {
805       \bool_lazy_or:nnTF
806         { \cs_if_exist_p:N \tikzpicture }
807         { \str_if_eq_p:nn { #1 } { standard } }
808         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
809         { \@@_error:n { bad-option-for-line-style } }
810     } ,
811   line-style .value_required:n = true ,
812   color .tl_set:N = \l_@@_xdots_color_tl ,
813   color .value_required:n = true ,
814   radius .code:n =
815     \hook_gput_code:nnn { begindocument } { . }
816     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
817   radius .value_required:n = true ,
818   inter .code:n =
819     \hook_gput_code:nnn { begindocument } { . }
820     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
821   radius .value_required:n = true ,

```

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

```

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

```

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

```

825   draw-first .code:n = \prg_do_nothing: ,
826   unknown .code:n = \@@_error:n { Unknown~key~for~xdots }
827 }

```

```

828 \keys_define:nn { nicematrix / rules }
829 {
830   color .tl_set:N = \l_@@_rules_color_tl ,
831   color .value_required:n = true ,
832   width .dim_set:N = \arrayrulewidth ,
833   width .value_required:n = true ,
834   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
835 }

```

```

836 \cs_new_protected:Npn \@@_err_key_color_inside:
837 {
838   \@@_error_or_warning:n { key~color~inside }
839   \cs_gset:Npn \@@_err_key_color_inside: { }
840 }

```

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

```

841 \keys_define:nn { nicematrix / Global }
842 {
843   color-inside .code:n = \@@_err_key_color_inside: ,
844   colortbl-like .code:n = \@@_err_key_color_inside: ,
845   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
846   ampersand-in-blocks .default:n = true ,
847   &-in-blocks .meta:n = ampersand-in-blocks ,
848   no-cell-nodes .code:n =
849     \bool_set_true:N \l_@@_no_cell_nodes_bool
850     \cs_set_protected:Npn \@@_node_cell:
851       { \set@color \box_use_drop:N \l_@@_cell_box } ,
852   no-cell-nodes .value_forbidden:n = true ,
853   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
854   rounded-corners .default:n = 4 pt ,
855   custom-line .code:n = \@@_custom_line:n { #1 } ,
856   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
857   rules .value_required:n = true ,
858   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
859   standard-cline .default:n = true ,
860   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
861   cell-space-top-limit .value_required:n = true ,
862   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
863   cell-space-bottom-limit .value_required:n = true ,
864   cell-space-limits .meta:n =
865     {
866       cell-space-top-limit = #1 ,
867       cell-space-bottom-limit = #1 ,
868     } ,
869   cell-space-limits .value_required:n = true ,
870   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
871   light-syntax .code:n =
872     \bool_set_true:N \l_@@_light_syntax_bool
873     \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
874   light-syntax .value_forbidden:n = true ,
875   light-syntax-expanded .code:n =
876     \bool_set_true:N \l_@@_light_syntax_bool
877     \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
878   light-syntax-expanded .value_forbidden:n = true ,
879   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
880   end-of-row .value_required:n = true ,
881   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
882   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
883   last-row .int_set:N = \l_@@_last_row_int ,
884   last-row .default:n = -1 ,
885   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
886   code-for-first-col .value_required:n = true ,
887   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
888   code-for-last-col .value_required:n = true ,
889   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
890   code-for-first-row .value_required:n = true ,
891   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
892   code-for-last-row .value_required:n = true ,
893   hlines .clist_set:N = \l_@@_hlines_clist ,
894   vlines .clist_set:N = \l_@@_vlines_clist ,
895   hlines .default:n = all ,

```

```

896 vlines .default:n = all ,
897 vlines-in-sub-matrix .code:n =
898 {
899   \tl_if_single_token:nTF { #1 }
900   {
901     \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
902     { \@@_error:nn { Forbidden-letter } { #1 } }

```

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

```

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

```

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

```

924 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
925 renew-dots .value_forbidden:n = true ,
926 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
927 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
928 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
929 create-extra-nodes .meta:n =
930 { create-medium-nodes , create-large-nodes } ,
931 left-margin .dim_set:N = \l_@@_left_margin_dim ,
932 left-margin .default:n = \arraycolsep ,
933 right-margin .dim_set:N = \l_@@_right_margin_dim ,
934 right-margin .default:n = \arraycolsep ,
935 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
936 margin .default:n = \arraycolsep ,
937 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
938 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
939 extra-margin .meta:n =
940 { extra-left-margin = #1 , extra-right-margin = #1 } ,
941 extra-margin .value_required:n = true ,
942 respect-arraystretch .code:n =
943   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
944 respect-arraystretch .value_forbidden:n = true ,
945 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
946 pgf-node-code .value_required:n = true
947 }

```

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

```

948 \keys_define:nn { nicematrix / environments }
949 {

```

```

950     corners .clist_set:N = \l_@@_corners_clist ,
951     corners .default:n = { NW , SW , NE , SE } ,
952     code-before .code:n =
953     {
954         \tl_if_empty:nF { #1 }
955         {
956             \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
957             \bool_set_true:N \l_@@_code_before_bool
958         }
959     } ,
960     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}`.

```

961     c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
962     t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
963     b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
964     baseline .tl_set:N = \l_@@_baseline_tl ,
965     baseline .value_required:n = true ,
966     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)`.

```

967     \str_if_eq:eeTF { #1 } { auto }
968     { \bool_set_true:N \l_@@_auto_columns_width_bool }
969     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
970     columns-width .value_required:n = true ,
971     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.

```

972     \legacy_if:nF { measuring@ }
973     {
974         \str_set:Ne \l_@@_name_str { #1 }
975         \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
976         { \@@_err_duplicate_names:n { #1 } }
977         { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
978     } ,
979     name .value_required:n = true ,
980     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
981     code-after .value_required:n = true ,
982 }

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

985 \keys_define:nn { nicematrix / notes }
986 {
987     para .bool_set:N = \l_@@_notes_para_bool ,
988     para .default:n = true ,
989     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
990     code-before .value_required:n = true ,
991     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
992     code-after .value_required:n = true ,
993     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
994     bottomrule .default:n = true ,
995     style .cs_set:Np = \@@_notes_style:n #1 ,
996     style .value_required:n = true ,
997     label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
998     label-in-tabular .value_required:n = true ,
999     label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1000     label-in-list .value_required:n = true ,
1001     enumitem-keys .code:n =
1002     {
1003         \hook_gput_code:nnn { begindocument } { . }

```



```

1004         {
1005             \IfPackageLoadedT { enumitem }
1006             { \setlist* [ tabularnotes ] { #1 } }
1007         }
1008     } ,
1009     enumitem-keys .value_required:n = true ,
1010     enumitem-keys-para .code:n =
1011     {
1012         \hook_gput_code:nnn { begindocument } { . }
1013         {
1014             \IfPackageLoadedT { enumitem }
1015             { \setlist* [ tabularnotes* ] { #1 } }
1016         }
1017     } ,
1018     enumitem-keys-para .value_required:n = true ,
1019     detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1020     detect-duplicates .default:n = true ,
1021     unknown .code:n = \@@_error:n { Unknown~key~for~notes }
1022 }
1023 \keys_define:nn { nicematrix / delimiters }
1024 {
1025     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1026     max-width .default:n = true ,
1027     color .tl_set:N = \l_@@_delimiters_color_tl ,
1028     color .value_required:n = true ,
1029 }

```

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

```

1030 \keys_define:nn { nicematrix }
1031 {
1032     NiceMatrixOptions .inherit:n =
1033     { nicematrix / Global } ,
1034     NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1035     NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1036     NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1037     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1038     SubMatrix / rules .inherit:n = nicematrix / rules ,
1039     CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1040     CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1041     CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1042     NiceMatrix .inherit:n =
1043     {
1044         nicematrix / Global ,
1045         nicematrix / environments ,
1046     } ,
1047     NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1048     NiceMatrix / rules .inherit:n = nicematrix / rules ,
1049     NiceTabular .inherit:n =
1050     {
1051         nicematrix / Global ,
1052         nicematrix / environments
1053     } ,
1054     NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1055     NiceTabular / rules .inherit:n = nicematrix / rules ,
1056     NiceTabular / notes .inherit:n = nicematrix / notes ,
1057     NiceArray .inherit:n =
1058     {
1059         nicematrix / Global ,
1060         nicematrix / environments ,
1061     } ,
1062     NiceArray / xdots .inherit:n = nicematrix / xdots ,

```

```

1063   NiceArray / rules .inherit:n = nicematrix / rules ,
1064   pNiceArray .inherit:n =
1065   {
1066       nicematrix / Global ,
1067       nicematrix / environments ,
1068   } ,
1069   pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1070   pNiceArray / rules .inherit:n = nicematrix / rules ,
1071 }

```

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

```

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

```

With the option `renew-matrix`, the environment `{matrix}` of `amsmath` and its variants are redefined to behave like the environment `{NiceMatrix}` and its variants.

```

1088   renew-matrix .code:n = \@@_renew_matrix: ,
1089   renew-matrix .value_forbidden:n = true ,

```

The option `exterior-arraycolsep` will have effect only in `{NiceArray}` for those who want to have for `{NiceArray}` the same behaviour as `{array}`.

```

1090   exterior-arraycolsep .bool_set:N = \l_@@_exterior_arraycolsep_bool ,

```

If the option `columns-width` is used, all the columns will have the same width.

In `\NiceMatrixOptions`, the special value `auto` is not available.

```

1091   columns-width .code:n =

```

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

```

1092       \str_if_eq:eeTF { #1 } { auto }
1093       { \@@_error:n { Option~auto~for~columns-width } }
1094       { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1095   allow-duplicate-names .code:n =
1096       \cs_set:Nn \@@_err_duplicate_names:n { } ,
1097   allow-duplicate-names .value_forbidden:n = true ,
1098   notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1099   notes .value_required:n = true ,
1100   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1101   sub-matrix .value_required:n = true ,
1102   matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1103   matrix / columns-type .value_required:n = true ,
1104   caption-above .bool_set:N = \l_@@_caption_above_bool ,

```

```

1105     caption-above .default:n = true ,
1106     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1107 }

```

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

```

1108 \NewDocumentCommand \NiceMatrixOptions { m }
1109 { \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.

```

1110 \keys_define:nn { nicematrix / NiceMatrix }
1111 {
1112     last-col .code:n = \tl_if_empty:nTF { #1 }
1113     {
1114         \bool_set_true:N \l_@@_last_col_without_value_bool
1115         \int_set:Nn \l_@@_last_col_int { -1 }
1116     }
1117     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1118     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1119     columns-type .value_required:n = true ,
1120     l .meta:n = { columns-type = l } ,
1121     r .meta:n = { columns-type = r } ,
1122     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1123     delimiters / color .value_required:n = true ,
1124     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1125     delimiters / max-width .default:n = true ,
1126     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1127     delimiters .value_required:n = true ,
1128     small .bool_set:N = \l_@@_small_bool ,
1129     small .value_forbidden:n = true ,
1130     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1131 }

```

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

```

1132 \keys_define:nn { nicematrix / NiceArray }
1133 {

```

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.

```

1134     small .bool_set:N = \l_@@_small_bool ,
1135     small .value_forbidden:n = true ,
1136     last-col .code:n = \tl_if_empty:nF { #1 }
1137     { \@@_error:n { last-col~non~empty~for~NiceArray } }
1138     \int_zero:N \l_@@_last_col_int ,
1139     r .code:n = \@@_error:n { r-or~l~with~preamble } ,
1140     l .code:n = \@@_error:n { r-or~l~with~preamble } ,
1141     unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1142 }
1143 \keys_define:nn { nicematrix / pNiceArray }
1144 {
1145     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1146     last-col .code:n = \tl_if_empty:nF { #1 }
1147     { \@@_error:n { last-col~non~empty~for~NiceArray } }
1148     \int_zero:N \l_@@_last_col_int ,
1149     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1150     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1151     delimiters / color .value_required:n = true ,
1152     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1153     delimiters / max-width .default:n = true ,

```

```

1154     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1155     delimiters .value_required:n = true ,
1156     small .bool_set:N = \l_@@_small_bool ,
1157     small .value_forbidden:n = true ,
1158     r .code:n = \@@_error:n { r-or~l-with~preamble } ,
1159     l .code:n = \@@_error:n { r-or~l-with~preamble } ,
1160     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1161 }

```

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

```

1162 \keys_define:nn { nicematrix / NiceTabular }
1163 {

```

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.

```

1164     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1165             \bool_set_true:N \l_@@_width_used_bool ,
1166     width .value_required:n = true ,
1167     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1168     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1169     tabularnote .value_required:n = true ,
1170     caption .tl_set:N = \l_@@_caption_tl ,
1171     caption .value_required:n = true ,
1172     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1173     short-caption .value_required:n = true ,
1174     label .tl_set:N = \l_@@_label_tl ,
1175     label .value_required:n = true ,
1176     last-col .code:n = \tl_if_empty:nF { #1 }
1177             { \@@_error:n { last-col-non-empty-for~NiceArray } }
1178             \int_zero:N \l_@@_last_col_int ,
1179     r .code:n = \@@_error:n { r-or~l-with~preamble } ,
1180     l .code:n = \@@_error:n { r-or~l-with~preamble } ,
1181     unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
1182 }

```

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
```

```

1183 \keys_define:nn { nicematrix / CodeAfter }
1184 {
1185     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1186     delimiters / color .value_required:n = true ,
1187     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1188     rules .value_required:n = true ,
1189     xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1190     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1191     sub-matrix .value_required:n = true ,
1192     unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1193 }

```

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}`).

```
1194 \cs_new_protected:Npn \@@_cell_begin:
1195 {
```

`\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.

```
1196 \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).

```
1197 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1198 \cs_set_eq:NN \Hline \@@_Hline_in_cell:
```

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

```
1199 \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.

```
1200 \int_compare:nNnT { \c@jCol } = { \c_one_int }
1201 {
1202   \int_compare:nNnT { \l_@@_first_col_int } = { \c_one_int }
1203   { \@@_begin_of_row: }
1204 }
```

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:`.

```
1205 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1206 \@@_tuning_not_tabular_begin:
1207 \@@_tuning_first_row:
1208 \@@_tuning_last_row:
1209 \g_@@_row_style_tl
1210 }
```

The following command will be nullified unless there is a first row.
Here is a version with the standard syntax of L3.

```
\cs_new_protected:Npn \@@_tuning_first_row:
{
  \int_if_zero:nT { \c@iRow }
  {
    \int_if_zero:nF { \c@jCol }
    {
      \l_@@_code_for_first_row_tl
      \xglobal \colorlet { nicematrix-first-row } { . }
    }
  }
}
```

We will use a version a little more efficient.

```

1211 \cs_new_protected:Npn \@@_tuning_first_row:
1212 {
1213   \if_int_compare:w \c@iRow = \c_zero_int
1214     \if_int_compare:w \c@jCol > \c_zero_int
1215       \l_@@_code_for_first_row_tl
1216       \xglobal \colorlet { nicematrix-first-row } { . }
1217     \fi:
1218   \fi:
1219 }

```

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

```

\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.

```

1220 \cs_new_protected:Npn \@@_tuning_last_row:
1221 {
1222   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1223     \l_@@_code_for_last_row_tl
1224     \xglobal \colorlet { nicematrix-last-row } { . }
1225   \fi:
1226 }

```

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

```

1227 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1228 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1229 {
1230   \m@th
1231   \c_math_toggle_token

```

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

```

1232   \@@_tuning_key_small:
1233 }
1234 \cs_set_eq:NN \@@_tuning_not_tabular_end: \c_math_toggle_token

```

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.

```

1235 \cs_new_protected:Npn \@@_begin_of_row:
1236 {
1237   \int_gincr:N \c@iRow
1238   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1239   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1240   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1241   \pgfpicture
1242   \pgfrememberpicturepositiononpagetrue
1243   \pgfcoordinate
1244     { \@@_env: - row - \int_use:N \c@iRow - base }
1245     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1246   \str_if_empty:NF \l_@@_name_str
1247     {
1248       \pgfnodelias
1249       { \l_@@_name_str - row - \int_use:N \c@iRow - base }

```

```

1250         { \@@_env: - row - \int_use:N \c@iRow - base }
1251     }
1252     \endpgfpicture
1253 }

```

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.

```

1254 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1255 {
1256     \int_if_zero:nTF { \c@iRow }
1257     {
1258         \dim_compare:nNnT
1259         { \box_dp:N \l_@@_cell_box } > { \g_@@_dp_row_zero_dim }
1260         { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1261         \dim_compare:nNnT
1262         { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_zero_dim }
1263         { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1264     }
1265     {
1266         \int_compare:nNnT { \c@iRow } = { \c_one_int }
1267         {
1268             \dim_compare:nNnT
1269             { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_one_dim }
1270             { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1271         }
1272     }
1273 }
1274 \cs_new_protected:Npn \@@_rotate_cell_box:
1275 {
1276     \box_rotate:Nn \l_@@_cell_box { 90 }
1277     \bool_if:NTF \g_@@_rotate_c_bool
1278     {
1279         \hbox_set:Nn \l_@@_cell_box
1280         {
1281             \m@th
1282             \c_math_toggle_token
1283             \vcenter { \box_use:N \l_@@_cell_box }
1284             \c_math_toggle_token
1285         }
1286     }
1287     {
1288         \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
1289         {
1290             \vbox_set_top:Nn \l_@@_cell_box
1291             {
1292                 \vbox_to_zero:n { }
1293                 \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1294                 \box_use:N \l_@@_cell_box
1295             }
1296         }
1297     }
1298     \bool_gset_false:N \g_@@_rotate_bool
1299     \bool_gset_false:N \g_@@_rotate_c_bool
1300 }
1301 \cs_new_protected:Npn \@@_adjust_size_box:
1302 {
1303     \dim_compare:nNnT { \g_@@_blocks_wd_dim } > { \c_zero_dim }
1304     {

```

```

1305     \box_set_wd:Nn \l_@@_cell_box
1306     { \dim_max:nn { \box_wd:N \l_@@_cell_box } { \g_@@_blocks_wd_dim } }
1307     \dim_gzero:N \g_@@_blocks_wd_dim
1308   }
1309   \dim_compare:nNnT { \g_@@_blocks_dp_dim } > { \c_zero_dim }
1310   {
1311     \box_set_dp:Nn \l_@@_cell_box
1312     { \dim_max:nn { \box_dp:N \l_@@_cell_box } { \g_@@_blocks_dp_dim } }
1313     \dim_gzero:N \g_@@_blocks_dp_dim
1314   }
1315   \dim_compare:nNnT { \g_@@_blocks_ht_dim } > { \c_zero_dim }
1316   {
1317     \box_set_ht:Nn \l_@@_cell_box
1318     { \dim_max:nn { \box_ht:N \l_@@_cell_box } { \g_@@_blocks_ht_dim } }
1319     \dim_gzero:N \g_@@_blocks_ht_dim
1320   }
1321 }
1322 \cs_new_protected:Npn \@@_cell_end:
1323 {

```

The following command is nullified in the tabulars.

```

1324   \@@_tuning_not_tabular_end:
1325   \hbox_set_end:
1326   \@@_cell_end_i:
1327 }
1328 \cs_new_protected:Npn \@@_cell_end_i:
1329 {

```

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.

```

1330   \g_@@_cell_after_hook_tl
1331   \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
1332   \@@_adjust_size_box:
1333   \box_set_ht:Nn \l_@@_cell_box
1334   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1335   \box_set_dp:Nn \l_@@_cell_box
1336   { \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”).

```

1337   \@@_update_max_cell_width:

```

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

```

1338   \@@_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.

```

1339 \bool_if:NTF \g_@@_empty_cell_bool
1340 { \box_use_drop:N \l_@@_cell_box }
1341 {
1342   \bool_if:NTF \g_@@_not_empty_cell_bool
1343   { \@@_print_node_cell: }
1344   {
1345     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
1346     { \@@_print_node_cell: }
1347     { \box_use_drop:N \l_@@_cell_box }
1348   }
1349 }
1350 \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
1351 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1352 \bool_gset_false:N \g_@@_empty_cell_bool
1353 \bool_gset_false:N \g_@@_not_empty_cell_bool
1354 }

```

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

```

1355 \cs_new_protected:Npn \@@_update_max_cell_width:
1356 {
1357   \dim_gset:Nn \g_@@_max_cell_width_dim
1358   { \dim_max:nn { \g_@@_max_cell_width_dim } { \box_wd:N \l_@@_cell_box } }
1359 }

```

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`).

```

1360 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1361 {
1362   \@@_math_toggle:
1363   \hbox_set_end:
1364   \bool_if:NF \g_@@_rotate_bool
1365   {
1366     \hbox_set:Nn \l_@@_cell_box
1367     {
1368       \makebox [ \l_@@_col_width_dim ] [ s ]
1369       { \hbox_unpack_drop:N \l_@@_cell_box }
1370     }
1371   }
1372   \@@_cell_end_i:
1373 }

```

```

1374 \pgfset
1375 {
1376   nicematrix / cell-node /.style =
1377   {
1378     inner~sep = \c_zero_dim ,
1379     minimum~width = \c_zero_dim
1380   }
1381 }

```

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.

```

1382 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1383 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1384 {
1385   \use:c
1386   {
1387     __siunitx_table_align_
1388     \bool_if:NTF \l__siunitx_table_text_bool
1389       { \l__siunitx_table_align_text_tl }
1390       { \l__siunitx_table_align_number_tl }
1391     :n
1392   }
1393   { #1 }
1394 }

```

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!

```

1395 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1396 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1397 {
1398   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1399   \hbox:n
1400   {
1401     \pgfsys@markposition
1402     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1403   }
1404   #1
1405   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1406   \hbox:n
1407   {
1408     \pgfsys@markposition
1409     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1410   }
1411 }

1412 \cs_new_protected:Npn \@@_print_node_cell:
1413 {
1414   \socket_use:nn { nicematrix / siunitx-wrap }
1415   { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1416 }

```

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

```

1417 \cs_new_protected:Npn \@@_node_cell:
1418 {
1419   \pgfpicture
1420   \pgfsetbaseline \c_zero_dim
1421   \pgfrememberpicturepositiononpagetrue
1422   \pgfset { nicematrix / cell-node }
1423   \pgfnode
1424   { rectangle }
1425   { base }
1426   {

```

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).

```

1427   \sys_if_engine_xetex:T { \set@color }
1428   \box_use:N \l_@@_cell_box
1429 }
1430 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1431 { \l_@@_pgf_node_code_tl }

```

```

1432 \str_if_empty:NF \l_@@_name_str
1433 {
1434   \pgfnodealias
1435     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1436     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1437 }
1438 \endpgfpicture
1439 }

```

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}

```

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

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).

```

1440 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1441 {
1442   \bool_if:nTF { #1 } { \tl_gput_left:ce } { \tl_gput_right:ce }
1443   { g_@@_ #2 _ lines _ tl }
1444   {
1445     \use:c { @@ _ draw _ #2 : nnn }
1446     { \int_use:N \c@iRow }
1447     { \int_use:N \c@jCol }
1448     { \exp_not:n { #3 } }
1449   }
1450 }

1451 \cs_new_protected:Npn \@@_array:n
1452 {
1453   \dim_set:Nn \col@sep
1454   { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1455   \dim_compare:nNnTF { \l_@@_tabular_width_dim } = { \c_zero_dim }
1456   { \def \@halignto { } }
1457   { \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`.

```

1458 \tabarray

```

`\l_@@_baseline_tl` may have the value `t`, `c` or `b`. However, if the value is `b`, we compose the `\array` (of array) with the option `t` and the right translation will be done further. Remark that `\str_if_eq:eeTF` is fully expandable and we need something fully expandable here. `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```

1459 [ \str_if_eq:eeTF { \l_@@_baseline_tl } { c } { c } { t } ]
1460 ]
1461 \cs_generate_variant:Nn \@@_array:n { o }

```

We keep in memory the standard version of `\ar@ialign` because we will redefine `\ialign` in the environment `{NiceArrayWithDelims}` but restore the standard version for use in the cells of the array. However, it seems that RevTeX goes on with a redefinition of `array` which uses `\ialign`.

```
1462 \bool_if:NTF \c_@@_revtex_bool
1463 { \cs_set_eq:NN \@@_old_ialign: \ialign }
```

We use here a `\cs_set_eq:cN` instead of a `\cs_set_eq:NN` in order to avoid a message when `explcheck` is used on `nicematrix.sty`.

```
1464 { \cs_set_eq:cN { \@@_old_ar@ialign: } \ar@ialign }
```

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

```
1465 \cs_new_protected:Npn \@@_create_row_node:
1466 {
1467   \int_compare:nNt { \c@iRow } > { \g_@@_last_row_node_int }
1468   {
1469     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1470     \@@_create_row_node_i:
1471   }
1472 }
1473 \cs_new_protected:Npn \@@_create_row_node_i:
1474 {
```

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

```
1475   \hbox
1476   {
1477     \bool_if:NT \l_@@_code_before_bool
1478     {
1479       \vtop
1480       {
1481         \skip_vertical:N 0.5\arrayrulewidth
1482         \pgfsys@markposition
1483         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1484         \skip_vertical:N -0.5\arrayrulewidth
1485       }
1486     }
1487     \pgfpicture
1488     \pgfrememberpicturepositiononpagetrue
1489     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1490     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1491     \str_if_empty:NF \l_@@_name_str
1492     {
1493       \pgfnodealias
1494       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1495       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1496     }
1497     \endpgfpicture
1498   }
1499 }
```

```
1500 \cs_new_protected:Npn \@@_in_everycr:
1501 {
1502   \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1503   \tbl_update_cell_data_for_next_row:
1504   \int_gzero:N \c@jCol
1505   \bool_gset_false:N \g_@@_after_col_zero_bool
1506   \bool_if:NF \g_@@_row_of_col_done_bool
1507   {
1508     \@@_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).

```
1509   \clist_if_empty:NF \l_@@_hlines_clist
1510   {
```

```

1511     \str_if_eq:eeF { \l_@@_hlines_clist } { all }
1512     {
1513         \clist_if_in:NeT
1514         \l_@@_hlines_clist
1515         { \int_eval:n { \c@iRow + 1 } }
1516     }
1517     {

```

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.

```

1518         \int_compare:nNnT { \c@iRow } > { -1 }
1519         {
1520             \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
1521             { \hrule height \arrayrulewidth width \c_zero_dim }
1522         }
1523     }
1524 }
1525 }
1526 }

```

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

```

1527 \cs_set_protected:Npn \@@_renew_dots:
1528 {
1529     \cs_set_eq:NN \ldots \@@_Ldots:
1530     \cs_set_eq:NN \cdots \@@_Cdots:
1531     \cs_set_eq:NN \vdots \@@_Vdots:
1532     \cs_set_eq:NN \ddots \@@_Ddots:
1533     \cs_set_eq:NN \iddots \@@_Iddots:
1534     \cs_set_eq:NN \dots \@@_Ldots:
1535     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1536 }

```

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.

```

1537 \hook_gput_code:nnn { begindocument } { . }
1538 {
1539     \IfPackageLoadedTF { booktabs }
1540     {
1541         \cs_new_protected:Npn \@@_patch_booktabs:
1542         { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1543     }
1544     { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1545 }

```

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`.

```

1546 \cs_new_protected:Npn \@@_some_initialization:
1547 {

```

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}`.

```

1548 \@@_everycr:
1549 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1550 \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1551 \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1552 \dim_gzero:N \g_@@_dp_ante_last_row_dim
1553 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1554 \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1555 }

```

`\@@_pre_array_ii:` contains code executed after the analyse of the keys and after the execution of the `\CodeBefore`.

```

1556 \cs_new_protected:Npn \@@_pre_array_ii:
1557 {

```

The total weight of the letters X in the preamble of the array.

```

1558 \fp_gzero:N \g_@@_total_X_weight_fp
1559 \bool_gset_false:N \g_@@_V_of_X_bool
1560 \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1561 \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol
1562 \@@_patch_booktabs:
1563 \box_clear_new:N \l_@@_cell_box
1564 \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}`).

```

1565 \bool_if:NT \l_@@_small_bool
1566 {
1567     \def \arraystretch { 0.47 }
1568     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1569 \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1570 }

```

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!

```

1571 \bool_if:NT \g_@@_create_cell_nodes_bool
1572 {
1573     \tl_put_right:Nn \@@_begin_of_row:
1574     {
1575         \pgfsys@markposition
1576         { \@@_env: - row - \int_use:N \c@iRow - base }
1577     }
1578     \socket_assign_plugin { nicematrix / create-cell-nodes } { active }
1579 }

```

The environment `{array}` (since version 2.6) uses internally the command `\ar@ialign` (and previously, it was `\ialign`). We change that command for several reasons. In particular, `\ar@ialign` sets `\everycr` to `{ }` and we *need* to change the value of `\everycr`.

```

1580 \bool_if:NF \c_@@_revtex_bool
1581 {
1582     \def \ar@ialign
1583     {
1584         \IfPackageLoadedT { latex-lab-testphase-table }
1585         { \tbl_init_cell_data_for_table: }
1586         \@@_some_initialization:
1587         \dim_zero:N \tabskip

```

After its first use, the definition of `\ar@ialign` will revert automatically to its default definition. With this 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`.

```

1588         \cs_set_eq:Nc \ar@ialign { @@_old_ar@ialign: }
1589         \halign
1590     }
1591 }

```

It seems that there is a problem when `nicematrix` is used with in `revtex4-2` with the package `colortbl` loaded. The following code prevent that problem but it does *not* treat the actual problem! It's only a patch *ad hoc*.

That patch has been added in version 7.0x, 2024-11-27 (question by mail of Tamra Nebabu).

```

1592     \bool_if:NT \c_@@_revtex_bool
1593     {
1594         \IfPackageLoadedT { colortbl }
1595         { \cs_set_protected:Npn \CT@setup { } }
1596     }

```

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).

```

1597     \cs_set_eq:NN \@@_old_ldots: \ldots
1598     \cs_set_eq:NN \@@_old_cdots: \cdots
1599     \cs_set_eq:NN \@@_old_vdots: \vdots
1600     \cs_set_eq:NN \@@_old_ddots: \ddots
1601     \cs_set_eq:NN \@@_old_iddots: \iddots
1602     \bool_if:NTF \l_@@_standard_cline_bool
1603     { \cs_set_eq:NN \cline \@@_standard_cline: }
1604     { \cs_set_eq:NN \cline \@@_cline: }
1605     \cs_set_eq:NN \Ldots \@@_Ldots:
1606     \cs_set_eq:NN \Cdots \@@_Cdots:
1607     \cs_set_eq:NN \Vdots \@@_Vdots:
1608     \cs_set_eq:NN \Ddots \@@_Ddots:
1609     \cs_set_eq:NN \Iddots \@@_Iddots:
1610     \cs_set_eq:NN \Hline \@@_Hline:
1611     \cs_set_eq:NN \Hspace \@@_Hspace:
1612     \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1613     \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1614     \cs_set_eq:NN \Block \@@_Block:
1615     \cs_set_eq:NN \rotate \@@_rotate:
1616     \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1617     \cs_set_eq:NN \dotfill \@@_dotfill:
1618     \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1619     \cs_set_eq:NN \diagbox \@@_diagbox:nn
1620     \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1621     \cs_set_eq:NN \TopRule \@@_TopRule
1622     \cs_set_eq:NN \MidRule \@@_MidRule
1623     \cs_set_eq:NN \BottomRule \@@_BottomRule
1624     \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1625     \cs_set_eq:NN \Hbrace \@@_Hbrace
1626     \cs_set_eq:NN \Vbrace \@@_Vbrace
1627     \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1628     { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1629     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1630     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1631     \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1632     \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1633     \int_compare:nNnT { \l_@@_first_row_int } > { \c_zero_int }
1634     { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1635     \int_compare:nNnT { \l_@@_last_row_int } < { \c_zero_int }
1636     { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1637     \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:.`

```

1638 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1639 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1640 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1641 \@@_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!).

```

1642 \tl_if_exist:NT \l_@@_note_in_caption_tl
1643 {
1644   \tl_if_empty:NF \l_@@_note_in_caption_tl
1645   {
1646     \int_gset:Nn \g_@@_notes_caption_int { \l_@@_note_in_caption_tl }
1647     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1648   }
1649 }

```

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).

```

1650 \seq_gclear:N \g_@@_multicolumn_cells_seq
1651 \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).

```

1652 \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`).

```

1653 \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.

```

1654 \int_gzero:N \g_@@_col_total_int
1655 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1656 \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.

```

1657 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1658 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1659 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1660 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1661 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1662 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1663 \tl_gclear:N \g_nicematrix_code_before_tl
1664 \tl_gclear:N \g_@@_pre_code_before_tl
1665 }

```

This is the end of `\@@_pre_array_ii:.`

The command `\@@_pre_array:` will be executed after analyse of the keys of the environment.

```

1666 \cs_new_protected:Npn \@@_pre_array:
1667 {

```



```

1668 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1669 \int_gzero_new:N \c@iRow
1670 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1671 \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).

```

1672 \bool_if:NT \g_@@_aux_found_bool
1673 {
1674   \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1675   \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1676   \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1677   \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1678 }

```

We recall that `\l_@@_last_row_int` and `\l_@@_last_column_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-column` (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).

```

1679 \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
1680 {
1681   \bool_set_true:N \l_@@_last_row_without_value_bool
1682   \bool_if:NT \g_@@_aux_found_bool
1683     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1684 }
1685 \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
1686 {
1687   \bool_if:NT \g_@@_aux_found_bool
1688     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1689 }

```

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”.

```

1690 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
1691 {
1692   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1693     {
1694       \dim_compare:nNnT { \g_@@_ht_last_row_dim } < { \box_ht:N \l_@@_cell_box }
1695         { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1696       \dim_compare:nNnT { \g_@@_dp_last_row_dim } < { \box_dp:N \l_@@_cell_box }
1697         { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1698     }
1699 }

1700 \seq_gclear:N \g_@@_cols_vlism_seq
1701 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1702 \bool_if:NT \l_@@_code_before_bool { \@@_exec_code_before: }

```

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 clear that variable because it will be reconstructed during the creation of the array.

```

1703 \seq_gset_eq:NN \g_@@_pos_of_blocks_seq \g_@@_future_pos_of_blocks_seq
1704 \seq_gclear:N \g_@@_future_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```
1705 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1706 \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.

```
1707 \int_gset:Nn \g_@@_last_row_node_int { -2 }
```

The value -2 is important.

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

```
1708 \@@_pre_array_ii:
```

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

```
1709 \box_clear_new:N \l_@@_the_array_box
```

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]`).

```
1710 \dim_zero_new:N \l_@@_left_delim_dim
1711 \dim_zero_new:N \l_@@_right_delim_dim
1712 \bool_if:NTF \g_@@_delims_bool
1713 {
```

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

```
1714 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1715 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1716 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1717 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1718 }
1719 {
1720 \dim_gset:Nn \l_@@_left_delim_dim
1721 { 2 \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1722 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1723 }
```

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 `\c_math_toggle_token` also).

```
1724 \hbox_set:Nw \l_@@_the_array_box
1725 \skip_horizontal:N \l_@@_left_margin_dim
1726 \skip_horizontal:N \l_@@_extra_left_margin_dim
1727 \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).

```
1728 \m@th
1729 \c_math_toggle_token
1730 \bool_if:NTF \l_@@_light_syntax_bool
1731 { \use:c { @@-light-syntax } }
1732 { \use:c { @@-normal-syntax } }
1733 }
```

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

```

1734 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1735 {
1736   \tl_set:Nn \l_tmpa_tl { #1 }
1737   \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1738     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1739   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1740   \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...

```

1741   \@@_pre_array:
1742 }

```

9 The `\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).

```

1743 \cs_new_protected:Npn \@@_pre_code_before:
1744 {

```

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.

```

1745   \pgfsys@markposition { \@@_env: - position }
1746   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1747   \pgfpicture
1748   \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1749   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int + 1 }
1750   {
1751     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1752     \pgfcoordinate { \@@_env: - row - ##1 }
1753       { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1754   }

```

Now, the recreation of the `col` nodes.

```

1755   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int + 1 }
1756   {
1757     \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1758     \pgfcoordinate { \@@_env: - col - ##1 }
1759       { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1760   }

```

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

```

1761   \@@_create_diag_nodes:

```

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

```

1762   \bool_if:NT \g_@@_create_cell_nodes_bool { \@@_recreate_cell_nodes: }
1763   \endpgfpicture

```

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

```

1764   \@@_create_blocks_nodes:

```

```

1765 \IfPackageLoadedT { tikz }
1766 {
1767     \tikzset
1768     {
1769         every-picture / .style =
1770         { overlay , name-prefix = \@@_env: - }
1771     }
1772 }
1773 \cs_set_eq:NN \cellcolor \@@_cellcolor
1774 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1775 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1776 \cs_set_eq:NN \rowcolor \@@_rowcolor
1777 \cs_set_eq:NN \rowcolors \@@_rowcolors
1778 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1779 \cs_set_eq:NN \arraycolor \@@_arraycolor
1780 \cs_set_eq:NN \columncolor \@@_columncolor
1781 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1782 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1783 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1784 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1785 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1786 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1787 }

```

```

1788 \cs_new_protected:Npn \@@_exec_code_before:
1789 {

```

We mark the cells which are in the (empty) corners because those cells must not be colored. We should try to find a way to detected whether we actually have coloring instructions to execute...

```

1790 \clist_map_inline:Nn \l_@@_corners_cells_clist
1791 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1792 \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`.

```

1793 \@@_add_to_colors_seq:nn { { nocolor } } { }
1794 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1795 \group_begin:

```

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

```

1796 \if_mode_math:
1797     \@@_exec_code_before_i:
1798 \else:
1799     \c_math_toggle_token
1800     \@@_exec_code_before_i:
1801     \c_math_toggle_token
1802 \fi:
1803 \group_end:
1804 }

```

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`).

```

1805 \cs_new_protected:Npn \@@_exec_code_before_i:
1806 {
1807     \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1808     { \@@_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`.

```
1809 \exp_last_unbraced:No \@@_CodeBefore_keys:
1810 \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.

```
1811 \@@_actually_color:
1812 \l_@@_code_before_tl
1813 \q_stop
1814 }
```

```
1815 \keys_define:nn { nicematrix / CodeBefore }
1816 {
1817   create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1818   create-cell-nodes .default:n = true ,
1819   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1820   sub-matrix .value_required:n = true ,
1821   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1822   delimiters / color .value_required:n = true ,
1823   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1824 }

1825 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1826 {
1827   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1828   \@@_CodeBefore:w
1829 }
```

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.

```
1830 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1831 {
1832   \bool_if:NT \g_@@_aux_found_bool
1833   {
1834     \@@_pre_code_before:
1835     \legacy_if:nF { measuring@ } { #1 }
1836   }
1837 }
```

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.

```
1838 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1839 {
1840   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
1841   {
1842     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1843     \pgfcoordinate { \@@_env: - row - ##1 - base }
1844     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1845     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
1846     {
1847       \cs_if_exist:cT
1848       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - ####1 - NW }
1849       {
1850         \pgfsys@getposition
```

```

1851         { \@@_env: - ##1 - #####1 - NW }
1852         \@@_node_position:
1853         \pgfsys@getposition
1854         { \@@_env: - ##1 - #####1 - SE }
1855         \@@_node_position_i:
1856         \@@_pgf_rect_node:nnn
1857         { \@@_env: - ##1 - #####1 }
1858         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1859         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1860     }
1861 }
1862 }
1863 \@@_create_extra_nodes:
1864 \@@_create_aliases_last:
1865 }

1866 \cs_new_protected:Npn \@@_create_aliases_last:
1867 {
1868     \int_step_inline:nn { \c@iRow }
1869     {
1870         \pgfnodealias
1871         { \@@_env: - ##1 - last }
1872         { \@@_env: - ##1 - \int_use:N \c@jCol }
1873     }
1874     \int_step_inline:nn { \c@jCol }
1875     {
1876         \pgfnodealias
1877         { \@@_env: - last - ##1 }
1878         { \@@_env: - \int_use:N \c@iRow - ##1 }
1879     }
1880     \pgfnodealias
1881     { \@@_env: - last - last }
1882     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1883 }

1884 \cs_new_protected:Npn \@@_create_blocks_nodes:
1885 {
1886     \pgfpicture
1887     \pgf@relevantforpicturesizefalse
1888     \pgfrememberpicturepositiononpagetrue
1889     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1890     { \@@_create_one_block_node:nnnnn ##1 }
1891     \endpgfpicture
1892 }

```

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

```

1893 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1894 {
1895     \tl_if_empty:nF { #5 }
1896     {
1897         \@@_qpoint:n { col - #2 }
1898         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1899         \@@_qpoint:n { #1 }
1900         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1901         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1902         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1903         \@@_qpoint:n { \int_eval:n { #3 + 1 } }

```

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).

```

1904     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1905     \@@_pgf_rect_node:nnnnn
1906     { \@@_env: - #5 }
1907     { \dim_use:N \l_tmpa_dim }
1908     { \dim_use:N \l_tmpb_dim }
1909     { \dim_use:N \l_@@_tmpc_dim }
1910     { \dim_use:N \l_@@_tmpd_dim }
1911   }
1912 }

1913 \cs_new_protected:Npn \@@_patch_for_revtext:
1914 {
1915   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1916   \cs_set_eq:NN \@array \@array@array
1917   \cs_set_eq:NN \@tabular \@tabular@array
1918   \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1919   \cs_set_eq:NN \array \array@array
1920   \cs_set_eq:NN \endarray \endarray@array
1921   \cs_set:Npn \endtabular { \endarray $\egroup } % $
1922   \cs_set_eq:NN \@mkpream \@mkpream@array
1923   \cs_set_eq:NN \@classx \@classx@array
1924   \cs_set_eq:NN \insert@column \insert@column@array
1925   \cs_set_eq:NN \@arraycr \@arraycr@array
1926   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1927   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1928 }

```

10 The environment {NiceArrayWithDelims}

```

1929 \NewDocumentEnvironment { NiceArrayWithDelims }
1930 { m m O { } m ! O { } t \CodeBefore }
1931 {
1932   \bool_if:NT \c_@@_revtex_bool { \@@_patch_for_revtext: }
1933   \@@_provide_pgfsyspdfmark:
1934   \bool_if:NT \g_@@_footnote_bool { \savenotes }

```

The aim of the following `\bgroup` (the corresponding `\egroup` is, of course, at the end of the environment) is to be able to put an exposant to a matrix in a mathematical formula.

```

1935   \bgroup

1936   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1937   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1938   \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1939   \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

1940   \int_gzero:N \g_@@_block_box_int
1941   \dim_gzero:N \g_@@_width_last_col_dim
1942   \dim_gzero:N \g_@@_width_first_col_dim
1943   \bool_gset_false:N \g_@@_row_of_col_done_bool
1944   \str_if_empty:NT \g_@@_name_env_str
1945     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1946   \bool_if:NTF \l_@@_tabular_bool
1947     { \mode_leave_vertical: }
1948     { \@@_test_if_math_mode: }
1949   \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1950   \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.

```
1951 \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.

```
1952 \cs_if_exist:NT \tikz@library@external@loaded
1953 {
1954     \tikzexternaldisable
1955     \cs_if_exist:NT \ifstandalone
1956     { \tikzset { external / optimize = false } }
1957 }
```

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

```
1958 \int_gincr:N \g_@@_env_int
1959 \bool_if:NF \l_@@_block_auto_columns_width_bool
1960 { \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.

```
1961 \seq_gclear:N \g_@@_blocks_seq
1962 \seq_gclear:N \g_@@_pos_of_blocks_seq
```

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

```
1963 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1964 \seq_gclear:N \g_@@_pos_of_xdots_seq
1965 \tl_gclear_new:N \g_@@_code_before_tl
1966 \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.

```
1967 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1968 {
1969     \bool_gset_true:N \g_@@_aux_found_bool
1970     \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1971 }
1972 { \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.

```
1973 \tl_gclear:N \g_@@_aux_tl
1974 \tl_if_empty:NF \g_@@_code_before_tl
1975 {
1976     \bool_set_true:N \l_@@_code_before_bool
1977     \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1978 }
1979 \tl_if_empty:NF \g_@@_pre_code_before_tl
1980 { \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`.

```
1981 \bool_if:NTF \g_@@_delims_bool
1982 { \keys_set:nn { nicematrix / pNiceArray } }
1983 { \keys_set:nn { nicematrix / NiceArray } }
1984 { #3 , #5 }
```

⁸e.g. `\color[rgb]{0.5,0.5,0}`


```
1985 \@@_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:.

```
1986 \bool_if:nTF { #6 } { \@@_CodeBefore_Body:w } { \@@_pre_array: }
1987 }
```

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

```
1988 {
1989   \bool_if:NTF \l_@@_light_syntax_bool
1990   { \use:c { end @@-light-syntax } }
1991   { \use:c { end @@-normal-syntax } }
1992   \c_math_toggle_token
1993   \skip_horizontal:N \l_@@_right_margin_dim
1994   \skip_horizontal:N \l_@@_extra_right_margin_dim
1995   \hbox_set_end:
1996   \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.

```
1997 \bool_if:NT \l_@@_width_used_bool
1998 {
1999   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2000   { \@@_error_or_warning:n { width-without-X-columns } }
2001 }
```

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 .

```
2002 \fp_compare:nNnT { \g_@@_total_X_weight_fp } > { \c_zero_fp }
2003 { \@@_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).

```
2004 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
2005 {
2006   \bool_if:NF \l_@@_last_row_without_value_bool
2007   {
2008     \int_compare:nNnF { \l_@@_last_row_int } = { \c@iRow }
2009     {
2010       \@@_error:n { Wrong-last-row }
2011       \int_set_eq:NN \l_@@_last_row_int \c@iRow
2012     }
2013   }
2014 }
```

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

```
2015 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2016 \bool_if:NTF \g_@@_last_col_found_bool
2017 { \int_gdecr:N \c@jCol }
2018 {
2019   \int_compare:nNnT { \l_@@_last_col_int } > { -1 }
2020   { \@@_error:n { last-col-not-used } }
```

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

2021 }

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

```
2022        \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2023        \int_compare:nNnT { \l_@@_last_row_int } > { -1 }
2024        { \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. 92).

```
2025        \int_if_zero:nT { \l_@@_first_col_int }
2026        { \skip_horizontal:N \g_@@_width_first_col_dim }
```

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

```
2027        \bool_if:nTF { ! \g_@@_delims_bool }
2028        {
2029            \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
2030            { \@@_use_arraybox_with_notes_c: }
2031            {
2032                \str_if_eq:eeTF { \l_@@_baseline_tl } { b }
2033                { \@@_use_arraybox_with_notes_b: }
2034                { \@@_use_arraybox_with_notes: }
2035            }
2036        }
```

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).

```
2037        {
2038            \int_if_zero:nTF { \l_@@_first_row_int }
2039            {
2040                \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2041                \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2042            }
2043            { \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

```
2044        \int_compare:nNnTF { \l_@@_last_row_int } > { -2 }
2045        {
2046            \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2047            \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2048        }
2049        { \dim_zero:N \l_tmpb_dim }
2050        \hbox_set:Nn \l_tmpa_box
2051        {
2052            \m@th
2053            \c_math_toggle_token
2054            \@@_color:o \l_@@_delimiters_color_tl
2055            \exp_after:wN \left \g_@@_left_delim_tl
2056            \vcenter
2057            {
```

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.

```
2058            \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2059            \hbox
2060            {
2061                \bool_if:NTF \l_@@_tabular_bool
2062                { \skip_horizontal:n { - \tabcolsep } }
2063                { \skip_horizontal:n { - \arraycolsep } }
2064                \@@_use_arraybox_with_notes_c:
2065                \bool_if:NTF \l_@@_tabular_bool
```

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).

```

2066         { \skip_horizontal:n { - \tabcolsep } }
2067         { \skip_horizontal:n { - \arraycolsep } }
2068     }

```

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

```

2069         \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2070     }
2071     \exp_after:wN \right \g_@@_right_delim_tl
2072     \c_math_toggle_token
2073 }

```

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.

```

2074     \bool_if:NTF \l_@@_delimiters_max_width_bool
2075     {
2076         \@@_put_box_in_flow_bis:nn
2077         { \g_@@_left_delim_tl }
2078         { \g_@@_right_delim_tl }
2079     }
2080     \@@_put_box_in_flow:
2081 }

```

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

```

2082     \bool_if:NT \g_@@_last_col_found_bool
2083     { \skip_horizontal:N \g_@@_width_last_col_dim }
2084     \bool_if:NT \l_@@_preamble_bool
2085     {
2086         \int_compare:nNnT { \c@jCol } < { \g_@@_static_num_of_col_int }
2087         { \@@_err_columns_not_used: }
2088     }
2089     \@@_after_array:

```

The aim of the following `\egroup` (the corresponding `\bgroup` is, of course, at the beginning of the environment) is to be able to put an exposant to a matrix in a mathematical formula.

```

2090     \egroup

```

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

```

2091     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2092     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2093     \iow_now:Ne \@mainaux
2094     {
2095         \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2096         \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2097         { \exp_not:o \g_@@_aux_tl }
2098     }
2099     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

```

```

2100     \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2101 }

```

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

```

2102 \cs_new_protected:Npn \@@_err_columns_not_used:
2103 {
2104     \@@_warning:n { columns~not~used }
2105     \cs_gset:Npn \@@_err_columns_not_used: { }
2106 }

```

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 .

```

2107 \cs_new_protected:Npn \@@_compute_width_X:
2108 {
2109   \tl_gput_right:Ne \g_@@_aux_tl
2110   {
2111     \bool_set_true:N \l_@@_X_columns_aux_bool
2112     \dim_set:Nn \l_@@_X_columns_dim
2113     {

```

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`

```

2114     \bool_lazy_and:nnTF
2115     { \g_@@_V_of_X_bool }
2116     { \l_@@_X_columns_aux_bool }
2117     { \dim_use:N \l_@@_X_columns_dim }
2118     {
2119       \dim_compare:nNnTF
2120       {
2121         \dim_abs:n
2122         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2123       }
2124       <
2125       { 0.001 pt }
2126       { \dim_use:N \l_@@_X_columns_dim }
2127       {
2128         \dim_eval:n
2129         {
2130           \l_@@_X_columns_dim
2131           +
2132           \fp_to_dim:n
2133           {
2134             (
2135               \dim_eval:n
2136               { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2137             )
2138             / \fp_use:N \g_@@_total_X_weight_fp
2139           }
2140         }
2141       }
2142     }
2143   }
2144 }
2145 }

```

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`.

```

2146 \cs_new_protected:Npn \@@_transform_preamble:
2147 {
2148   \@@_transform_preamble_i:
2149   \@@_transform_preamble_ii:
2150 }
2151 \cs_new_protected:Npn \@@_transform_preamble_i:
2152 {
2153   \int_gzero:N \c@jCol

```

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

```
2154 \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.

```
2155 \bool_gset_false:N \g_tmpb_bool
```

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

```
2156 \tl_gclear_new:N \g_@@_pre_cell_tl
```

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

```
2157 \int_zero:N \l_tmpa_int
2158 \tl_gclear:N \g_@@_array_preamble_tl
2159 \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2160 {
2161   \tl_gset:Nn \g_@@_array_preamble_tl
2162     { ! { \skip_horizontal:N \arrayrulewidth } }
2163 }
2164 {
2165   \clist_if_in:NnT \l_@@_vlines_clist 1
2166   {
2167     \tl_gset:Nn \g_@@_array_preamble_tl
2168       { ! { \skip_horizontal:N \arrayrulewidth } }
2169   }
2170 }
```

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

```
2171 \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2172 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2173 \@@_replace_columncolor:
2174 }

2175 \cs_new_protected:Npn \@@_transform_preamble_ii:
2176 {
```

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

```
2177 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2178 {
2179   \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2180   { \bool_gset_true:N \g_@@_delims_bool }
2181 }
2182 { \bool_gset_true:N \g_@@_delims_bool }
```

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

```
2183 \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).

```
2184 \int_if_zero:nTF { \l_@@_first_col_int }
2185 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2186 {
2187   \bool_if:NF \g_@@_delims_bool
2188   {
2189     \bool_if:NF \l_@@_tabular_bool
2190     {
2191       \clist_if_empty:NT \l_@@_vlines_clist
2192       {
2193         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2194         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
```

```

2195     }
2196   }
2197 }
2198 }
2199 \int_compare:nNnTF { \l_@@_last_col_int } > { -1 }
2200 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2201 {
2202   \bool_if:NF \g_@@_delims_bool
2203   {
2204     \bool_if:NF \l_@@_tabular_bool
2205     {
2206       \clist_if_empty:NT \l_@@_vlines_clist
2207       {
2208         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2209         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2210       }
2211     }
2212   }
2213 }

```

We add a last column to raise a good error message when the user puts more columns than allowed by its preamble. However, for technical reasons, it's not possible to do that in `{NiceTabular*}` (we control that with the value of `\l_@@_tabular_width_dim`).

```

2214 \dim_compare:nNnT { \l_@@_tabular_width_dim } = { \c_zero_dim }
2215 {

```

If the tagging of the tabulars is done (part of the Tagging Project), we don't activate that mechanism because it would create a dummy column of tagged empty cells.

```

2216   \IfPackageLoadedF { latex-lab-testphase-table }
2217   {
2218     \tl_gput_right:Nn \g_@@_array_preamble_tl
2219     { > { \@@_error_too_much_cols: } 1 }
2220   }
2221 }
2222 }

```

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.

```

2223 \cs_new_protected:Npn \@@_rec_preamble:n #1
2224 {

```

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

```

2225   \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2226   { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2227   {

```

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

```

2228   \cs_if_exist:cTF { NC @ find @ #1 }
2229   {
2230     \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2231     \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2232   }
2233   {
2234     \str_if_eq:nnTF { #1 } { S }
2235     { \@@_fatal:n { unknown~column~type~S } }
2236     { \@@_fatal:nn { unknown~column~type } { #1 } }
2237   }

```

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`.

```

2238     }
2239 }

```

For c, l and r

```

2240 \cs_new_protected:Npn \@@_c: #1
2241 {
2242   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2243   \tl_gclear:N \g_@@_pre_cell_tl
2244   \tl_gput_right:Nn \g_@@_array_preamble_tl
2245     { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2246   \int_gincr:N \c@jCol
2247   \@@_rec_preamble_after_col:n
2248 }
2249 \cs_new_protected:Npn \@@_l: #1
2250 {
2251   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2252   \tl_gclear:N \g_@@_pre_cell_tl
2253   \tl_gput_right:Nn \g_@@_array_preamble_tl
2254     {
2255       > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2256       l
2257       < \@@_cell_end:
2258     }
2259   \int_gincr:N \c@jCol
2260   \@@_rec_preamble_after_col:n
2261 }
2262 \cs_new_protected:Npn \@@_r: #1
2263 {
2264   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2265   \tl_gclear:N \g_@@_pre_cell_tl
2266   \tl_gput_right:Nn \g_@@_array_preamble_tl
2267     {
2268       > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2269       r
2270       < \@@_cell_end:
2271     }
2272   \int_gincr:N \c@jCol
2273   \@@_rec_preamble_after_col:n
2274 }

```

For ! and @

```

2275 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2276 {
2277   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2278   \@@_rec_preamble:n
2279 }
2280 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2281 \cs_new_protected:cpn { @@ _ | : } #1
2282 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2283   \int_incr:N \l_tmpa_int
2284   \@@_make_preamble_i_i:n
2285 }
2286 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2287 {

```

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

```

2288 \str_if_eq:nnTF { #1 } { | }
2289 { \use:c { @@ _ | : } | }
2290 { \@@_make_preamble_i_ii:nn { } #1 }
2291 }

2292 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2293 {
2294   \str_if_eq:nnTF { #2 } { [ ] }
2295   { \@@_make_preamble_i_ii:nw { #1 } [ ] }
2296   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2297 }
2298 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2299 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2300 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2301 {
2302   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2303   \tl_gput_right:Ne \g_@@_array_preamble_tl
2304   {

```

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

```

2305   \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2306 }
2307 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2308 {
2309   \@@_vline:n
2310   {
2311     position = \int_eval:n { \c@jCol + 1 } ,
2312     multiplicity = \int_use:N \l_tmpa_int ,
2313     total-width = \dim_use:N \l_@@_rule_width_dim ,
2314     #2
2315   }

```

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.

```

2316   }
2317   \int_zero:N \l_tmpa_int
2318   \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2319   \@@_rec_preamble:n #1
2320 }

2321 \cs_new_protected:cpn { @@ _ > : } #1 #2
2322 {
2323   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2324   \@@_rec_preamble:n
2325 }

2326 \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.

```

2327 \keys_define:nn { nicematrix / p-column }
2328 {
2329   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2330   r .value_forbidden:n = true ,
2331   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2332   c .value_forbidden:n = true ,
2333   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2334   l .value_forbidden:n = true ,
2335   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2336   S .value_forbidden:n = true ,
2337   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2338   p .value_forbidden:n = true ,

```



```

2339     t .meta:n = p ,
2340     m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2341     m .value_forbidden:n = true ,
2342     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2343     b .value_forbidden:n = true
2344 }

```

For p but also b and m.

```

2345 \cs_new_protected:Npn \@@_p: #1
2346 {
2347   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2348   \@@_make_preamble_ii_i:n
2349 }
2350 \cs_set_eq:NN \@@_b: \@@_p:
2351 \cs_set_eq:NN \@@_m: \@@_p:
2352 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2353 {
2354   \str_if_eq:nnTF { #1 } { [ ]
2355     { \@@_make_preamble_ii_ii:w [ ]
2356       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2357     }
2358   \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2359     { \@@_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.

```

2360 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2361 {

```

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).

```

2362   \str_set:Nn \l_@@_hpos_col_str { j }
2363   \@@_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`.

```

2364   \setlength { \l_tmpa_dim } { #2 }
2365   \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2366 }
2367 \cs_new_protected:Npn \@@_keys_p_column:n #1
2368 { \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.

```

2369 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2370 {

```

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

```

2371   \use:e
2372   {
2373     \@@_make_preamble_ii_vi:nnnnnnnn
2374     { \str_if_eq:eeTF { \l_@@_vpos_col_str } { p } { t } { b } }
2375     { #1 }
2376   }

```

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.

```

2377         \str_if_eq:eeTF { \l_@@_hpos_col_str } { j }
2378         { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2379         {

```

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

```

2380         \def \exp_not:N \l_@@_hpos_cell_tl
2381         { \str_lowercase:f { \l_@@_hpos_col_str } }
2382     }
2383     \IfPackageLoadedTF { ragged2e }
2384     {
2385         \str_case:on \l_@@_hpos_col_str
2386         {

```

The following `\exp_not:N` are mandatory.

```

2387             c { \exp_not:N \Centering }
2388             l { \exp_not:N \RaggedRight }
2389             r { \exp_not:N \RaggedLeft }
2390         }
2391     }
2392     {
2393         \str_case:on \l_@@_hpos_col_str
2394         {
2395             c { \exp_not:N \centering }
2396             l { \exp_not:N \raggedright }
2397             r { \exp_not:N \raggedleft }
2398         }
2399     }
2400     #3
2401 }
2402 { \str_if_eq:eeT { \l_@@_vpos_col_str } { m } \@@_center_cell_box: }
2403 { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_begin:w }
2404 { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_end: }
2405 { #2 }
2406 {
2407     \str_case:onF \l_@@_hpos_col_str
2408     {
2409         { j } { c }
2410         { si } { c }
2411     }

```

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

```

2412         { \str_lowercase:f \l_@@_hpos_col_str }
2413     }
2414 }

```

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

```

2415     \int_gincr:N \c@jCol
2416     \@@_rec_preamble_after_col:n
2417 }

```

#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*.

```

2418 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2419 {
2420   \str_if_eq:eeTF { \l_@@_hpos_col_str } { si }
2421   {
2422     \tl_gput_right:Nn \g_@@_array_preamble_tl
2423     { > \@@_test_if_empty_for_S: }
2424   }
2425   { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2426   \tl_gput_right:Nn \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2427   \tl_gclear:N \g_@@_pre_cell_tl
2428   \tl_gput_right:Nn \g_@@_array_preamble_tl
2429   {
2430     > {

```

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.

```

2431       \dim_set:Nn \l_@@_col_width_dim { #2 }
2432       \IfPackageLoadedT { latex-lab-testphase-table }
2433       { \tag_struct_begin:n { tag = Div } }
2434       \@@_cell_begin:

```

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

```

2435       \use:c { #7 } [ #1 ] { #2 }

```

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

```

2436       \everypar
2437       {
2438         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2439         \everypar { }
2440       }
2441       \IfPackageLoadedT { latex-lab-testphase-table }
2442       { \tagpdfparaOn }

```

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

```

2443       #3

```

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

```

2444       \g_@@_row_style_tl
2445       \arraybackslash
2446       #5
2447     }
2448     #8
2449     < {
2450       #6

```

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

```

2451       \finalstrut \@arstrutbox
2452       \use:c { end #7 }

```

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

```

2453       #4
2454       \@@_cell_end:
2455       \IfPackageLoadedT { latex-lab-testphase-table }
2456       { \tag_struct_end: }
2457     }
2458   }
2459 }

```

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

```
2460 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2461 {
```

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 `&`).

```
2462 \group_align_safe_begin:
2463 \peek_meaning:NTF &
2464 { \@@_the_cell_is_empty: }
2465 {
2466   \peek_meaning:NTF \\\
2467   { \@@_the_cell_is_empty: }
2468   {
2469     \peek_meaning:NTF \crcr
2470     \@@_the_cell_is_empty:
2471     \group_align_safe_end:
2472   }
2473 }
2474 }

2475 \cs_new_protected:Npn \@@_the_cell_is_empty:
2476 {
2477   \group_align_safe_end:
2478   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2479   {
```

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

```
2480   \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2481   \skip_horizontal:N \l_@@_col_width_dim
2482 }
2483 }

2484 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2485 {
2486   \peek_meaning:NT \__siunitx_table_skip:n
2487   { \bool_gset_true:N \g_@@_empty_cell_bool }
2488 }
```

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.

```
2489 \cs_new_protected:Npn \@@_center_cell_box:
2490 {
```

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

```
2491   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2492   {
2493     \dim_compare:nNnT
2494     { \box_ht:N \l_@@_cell_box }
2495     >
```

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).

```
2496     { \box_ht:N \strutbox }
2497     {
2498       \hbox_set:Nn \l_@@_cell_box
2499       {
```

```

2500         \box_move_down:nn
2501         {
2502             ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2503               + \baselineskip ) / 2
2504         }
2505         { \box_use:N \l_@@_cell_box }
2506     }
2507 }
2508 }
2509 }

```

For V (similar to the V of varwidth).

```

2510 \cs_new_protected:Npn \@@_V: #1 #2
2511 {
2512     \str_if_eq:nnTF { #2 } { [ ] }
2513     { \@@_make_preamble_V_i:w [ ] }
2514     { \@@_make_preamble_V_i:w [ ] { #2 } }
2515 }
2516 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2517 { \@@_make_preamble_V_ii:nn { #1 } }
2518 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2519 {
2520     \str_set:Nn \l_@@_vpos_col_str { p }
2521     \str_set:Nn \l_@@_hpos_col_str { j }
2522     \@@_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`.

```

2523 \setlength { \l_tmpa_dim } { #2 }
2524 \IfPackageLoadedTF { varwidth }
2525 { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2526 {
2527     \@@_error_or_warning:n { varwidth-not-loaded }
2528     \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2529 }
2530 }

```

For w and W

```

2531 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2532 \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.

```

2533 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2534 {
2535     \str_if_eq:nnTF { #3 } { s }
2536     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2537     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2538 }

```

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.

```

2539 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2540 {
2541     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2542     \tl_gclear:N \g_@@_pre_cell_tl
2543     \tl_gput_right:Nn \g_@@_array_preamble_tl

```

```

2544 {
2545   > {

```

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`.

```

2546     \setlength { \l_@@_col_width_dim } { #2 }
2547     \@@_cell_begin:
2548     \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2549   }
2550   c
2551   < {
2552     \@@_cell_end_for_w_s:
2553     #1
2554     \@@_adjust_size_box:
2555     \box_use_drop:N \l_@@_cell_box
2556   }
2557 }
2558 \int_gincr:N \c_jCol
2559 \@@_rec_preamble_after_col:n
2560 }

```

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

```

2561 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2562 {
2563   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2564   \tl_gclear:N \g_@@_pre_cell_tl
2565   \tl_gput_right:Nn \g_@@_array_preamble_tl
2566   {
2567     > {

```

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`.

```

2568     \setlength { \l_@@_col_width_dim } { #4 }
2569     \hbox_set:Nw \l_@@_cell_box
2570     \@@_cell_begin:
2571     \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2572   }
2573   c
2574   < {
2575     \@@_cell_end:
2576     \hbox_set_end:
2577     #1
2578     \@@_adjust_size_box:
2579     \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2580   }
2581 }

```

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

```

2582   \int_gincr:N \c_jCol
2583   \@@_rec_preamble_after_col:n
2584 }

```

```

2585 \cs_new_protected:Npn \@@_special_W:
2586 {
2587   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \l_@@_col_width_dim }
2588   { \@@_warning:n { W-warning } }
2589 }

```

For S (of siunitx).

```

2590 \cs_new_protected:Npn \@@_S: #1 #2
2591 {
2592   \str_if_eq:nnTF { #2 } { [ ] }
2593   { \@@_make_preamble_S:w [ ] }
2594   { \@@_make_preamble_S:w [ ] { #2 } }
2595 }
2596 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2597 { \@@_make_preamble_S_i:n { #1 } }
2598 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2599 {
2600   \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2601   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2602   \tl_gclear:N \g_@@_pre_cell_tl
2603   \tl_gput_right:Nn \g_@@_array_preamble_tl
2604   {
2605     > {

```

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).

```

2606   \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2607   \keys_set:nn { siunitx } { #1 }
2608   \@@_cell_begin:
2609   \siunitx_cell_begin:w
2610 }
2611 c
2612 <
2613 {
2614   \siunitx_cell_end:

```

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

```

2615   \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2616   {
2617     \bool_if:NTF \l__siunitx_table_text_bool
2618     { \bool_set_true:N }
2619     { \bool_set_false:N }
2620     \l__siunitx_table_text_bool
2621   }
2622   \@@_cell_end:
2623 }
2624 }

```

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

```

2625   \int_gincr:N \c@jCol
2626   \@@_rec_preamble_after_col:n
2627 }

```

For (, [and \{.

```

2628 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2629 {
2630   \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.

```

2631   \int_if_zero:nTF { \c@jCol }
2632   {
2633     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2634     {

```

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

```

2635         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2636         \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2637         \@@_rec_preamble:n #2
2638     }
2639     {
2640         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2641         \@@_make_preamble_iv:nn { #1 } { #2 }
2642     }
2643 }
2644 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2645 }
2646 \cs_set_eq:cc { @@ _ \token_to_str:N [ : ] { @@ _ \token_to_str:N ( : }
2647 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2648 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2649 {
2650     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2651     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2652     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2653     {
2654         \@@_error:nn { delimiter~after~opening } { #2 }
2655         \@@_rec_preamble:n
2656     }
2657     { \@@_rec_preamble:n #2 }
2658 }

```

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

```

2659 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2660 { \use:c { @@ _ \token_to_str:N ( : } }

```

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

```

2661 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2662 {
2663     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2664     \tl_if_in:nnTF { ) ] \} } { #2 }
2665     { \@@_make_preamble_v:nnn #1 #2 }
2666     {
2667         \str_if_eq:nnTF { \s_stop } { #2 }
2668         {
2669             \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2670             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2671             {
2672                 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2673                 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2674                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2675                 \@@_rec_preamble:n #2
2676             }
2677         }
2678         {
2679             \tl_if_in:nnT { ( [ \{ \left } { #2 }
2680             { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2681             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2682             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2683             \@@_rec_preamble:n #2
2684         }
2685     }
2686 }
2687 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2688 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }

```



```

2689 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2690 {
2691   \str_if_eq:nnTF { \s_stop } { #3 }
2692   {
2693     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2694     {
2695       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2696       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2697       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2698       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2699     }
2700     {
2701       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2702       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2703       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2704       \@@_error:nn { double~closing~delimiter } { #2 }
2705     }
2706   }
2707   {
2708     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2709     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2710     \@@_error:nn { double~closing~delimiter } { #2 }
2711     \@@_rec_preamble:n #3
2712   }
2713 }

2714 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2715 { \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 @{...}.

```

2716 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2717 {
2718   \str_if_eq:nnTF { #1 } { < }
2719   { \@@_rec_preamble_after_col_i:n }
2720   {
2721     \str_if_eq:nnTF { #1 } { @ }
2722     { \@@_rec_preamble_after_col_ii:n }
2723     {
2724       \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2725       {
2726         \tl_gput_right:Nn \g_@@_array_preamble_tl
2727         { ! { \skip_horizontal:N \arrayrulewidth } }
2728       }
2729       {
2730         \clist_if_in:NeT \l_@@_vlines_clist
2731         { \int_eval:n { \c@jCol + 1 } }
2732         {
2733           \tl_gput_right:Nn \g_@@_array_preamble_tl
2734           { ! { \skip_horizontal:N \arrayrulewidth } }
2735         }
2736       }
2737       \@@_rec_preamble:n { #1 }
2738     }
2739   }
2740 }

2741 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2742 {
2743   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2744   \@@_rec_preamble_after_col:n
2745 }

```

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.

```

2746 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2747 {
2748   \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2749   {
2750     \tl_gput_right:Nn \g_@@_array_preamble_tl
2751     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2752   }
2753   {
2754     \clist_if_in:NcTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2755     {
2756       \tl_gput_right:Nn \g_@@_array_preamble_tl
2757       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2758     }
2759     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2760   }
2761   \@@_rec_preamble:n
2762 }

2763 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2764 {
2765   \tl_clear:N \l_tmpa_tl
2766   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2767   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2768 }

```

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

```

2769 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2770 { \@@_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.

```

2771 \cs_new_protected:Npn \@@_X: #1 #2
2772 {
2773   \str_if_eq:nnTF { #2 } { [ ]
2774     { \@@_make_preamble_X:w [ ] }
2775     { \@@_make_preamble_X:w [ ] #2 }
2776   }
2777   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2778   { \@@_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`.

```

2779 \keys_define:nn { nicematrix / X-column }
2780 {
2781   V .code:n =
2782     \IfPackageLoadedTF { varwidth }
2783     {
2784       \bool_set_true:N \l_@@_V_of_X_bool
2785       \bool_gset_true:N \g_@@_V_of_X_bool
2786     }
2787     { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
2788   unknown .code:n =
2789     \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2790     { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }

```

```

2791         { \@@_error_or_warning:n { invalid-weight } }
2792     }

```

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

```

2793 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2794 {

```

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).

```

2795     \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).

```

2796     \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.

```

2797     \fp_set:Nn \l_tmpa_fp { 1.0 }

```

```

2798     \@@_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 now in the set of keys nicematrix/X-column in order to retrieve the potential weight explicitly provided by the final user.

```

2799     \bool_set_false:N \l_@@_V_of_X_bool
2800     \keys_set:no { nicematrix / X-column } \l_tmpa_tl

```

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

```

2801     \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).

```

2802     \bool_if:NTF \l_@@_X_columns_aux_bool
2803     {
2804         \@@_make_preamble_ii_iv:nnn

```

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

```

2805         { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2806         { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
2807         { \@@_no_update_width: }
2808     }

```

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.

```

2809     {
2810         \tl_gput_right:Nn \g_@@_array_preamble_tl
2811         {
2812             > {
2813                 \@@_cell_begin:
2814                 \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.

```

2815         \NotEmpty

```

The following code will nullify the box of the cell.

```

2816         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2817         { \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`.

```

2818         \begin { minipage } { 5 cm } \arraybackslash
2819     }
2820     c
2821     < {
2822         \end { minipage }
2823         \@@_cell_end:
2824     }
2825 }
2826 \int_gincr:N \c@jCol
2827 \@@_rec_preamble_after_col:n
2828 }
2829 }

2830 \cs_new_protected:Npn \@@_no_update_width:
2831 {
2832     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2833     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2834 }

```

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

```

2835 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2836 {
2837     \seq_gput_right:Ne \g_@@_cols_vlism_seq
2838     { \int_eval:n { \c@jCol + 1 } }
2839     \tl_gput_right:Ne \g_@@_array_preamble_tl
2840     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2841     \@@_rec_preamble:n
2842 }

```

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.

```

2843 \cs_set_eq:cN { @@ _ \token_to_str:N \s_stop : } \use_none:n

```

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

```

2844 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2845 { \@@_fatal:n { Preamble-forgotten } }
2846 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
2847 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
2848 { @@ _ \token_to_str:N \hline : }
2849 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
2850 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
2851 { @@ _ \token_to_str:N \hline : }
2852 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
2853 { @@ _ \token_to_str:N \hline : }
2854 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
2855 { @@ _ \token_to_str:N \hline : }
2856 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
2857 { @@ _ \token_to_str:N \hline : }

```

12 The redefinition of `\multicolumn`

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

```

2858 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2859 {

```

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`.

```

2860 \multispan { #1 }
2861 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2862 \begingroup
2863 \IfPackageLoadedTF { latex-lab-testphase-table }
2864   { \tbl_update_multicolumn_cell_data:n { #1 } }
2865 \def \@@addamp
2866   { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2867 \tl_gclear:N \g_@@_preamble_tl
2868 \@@_make_m_preamble:n #2 \q_stop

```

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

```

2869 \exp_args:No \mkpream \g_@@_preamble_tl
2870 \addtopreamble \@empty
2871 \endgroup
2872 \UseTaggingSocket { tbl / colspan } { #1 }

```

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

```

2873 \int_compare:nNnT { #1 } > { \c_one_int }
2874 {
2875   \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2876     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2877   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2878   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2879     {
2880       {
2881         \int_if_zero:nTF { \c@jCol }
2882           { \int_eval:n { \c@iRow + 1 } }
2883           { \int_use:N \c@iRow }
2884       }
2885       { \int_eval:n { \c@jCol + 1 } }
2886       {
2887         \int_if_zero:nTF { \c@jCol }
2888           { \int_eval:n { \c@iRow + 1 } }
2889           { \int_use:N \c@iRow }
2890       }
2891       { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block

```

2892     { }
2893   }
2894 }

```

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

```

2895 \RenewDocumentCommand { \cellcolor } { 0 { } m }
2896 {
2897   \tl_gput_right:Ne \g_@@_pre_code_before_tl
2898     {
2899       \@@_rectanglecolor [ ##1 ]
2900       { \exp_not:n { ##2 } }
2901       { \int_use:N \c@iRow - \int_use:N \c@jCol }
2902       { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2903     }
2904   \ignorespaces
2905 }

```

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

```
2906 \def \@sharp { #3 }
2907 \@arstrut
2908 \@preamble
2909 \null
```

We add some lines.

```
2910 \int_gadd:Nn \c@jCol { #1 - 1 }
2911 \int_compare:nNt { \c@jCol } > { \g_@@_col_total_int }
2912 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2913 \ignorespaces
2914 }
```

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`.

```
2915 \cs_new_protected:Npn \@_make_m_preamble:n #1
2916 {
2917   \str_case:nnF { #1 }
2918   {
2919     c { \@_make_m_preamble_i:n #1 }
2920     l { \@_make_m_preamble_i:n #1 }
2921     r { \@_make_m_preamble_i:n #1 }
2922     > { \@_make_m_preamble_ii:nn #1 }
2923     ! { \@_make_m_preamble_ii:nn #1 }
2924     @ { \@_make_m_preamble_ii:nn #1 }
2925     | { \@_make_m_preamble_iii:n #1 }
2926     p { \@_make_m_preamble_iv:nnn t #1 }
2927     m { \@_make_m_preamble_iv:nnn c #1 }
2928     b { \@_make_m_preamble_iv:nnn b #1 }
2929     w { \@_make_m_preamble_v:nnnn { } #1 }
2930     W { \@_make_m_preamble_v:nnnn { \@_special_W: } #1 }
2931     \q_stop { }
2932   }
2933   {
2934     \cs_if_exist:cTF { NC @ find @ #1 }
2935     {
2936       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2937       \exp_last_unbraced:No \@_make_m_preamble:n \l_tmpa_tl
2938     }
2939     {
2940       \str_if_eq:nnTF { #1 } { S }
2941       { \@_fatal:n { unknown~column~type~S~multicolumn } }
2942       { \@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
2943     }
2944   }
2945 }
```

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

```
2946 \cs_new_protected:Npn \@_make_m_preamble_i:n #1
2947 {
2948   \tl_gput_right:Nn \g_@@_preamble_tl
2949   {
2950     > { \@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
2951     #1
2952     < \@_cell_end:
2953   }
```

We test for the presence of a `<`.

```
2954 \@_make_m_preamble_x:n
2955 }
```

For >, ! and @

```

2956 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2957 {
2958   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2959   \@@_make_m_preamble:n
2960 }

```

For |

```

2961 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2962 {
2963   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2964   \@@_make_m_preamble:n
2965 }

```

For p, m and b

```

2966 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2967 {
2968   \tl_gput_right:Nn \g_@@_preamble_tl
2969   {
2970     > {
2971       \@@_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`.

```

2972       \setlength { \l_tmpa_dim } { #3 }
2973       \begin { minipage } [ #1 ] { \l_tmpa_dim }
2974       \mode_leave_vertical:
2975       \arraybackslash
2976       \vrule height \box_ht:N \@@arstrutbox depth \c_zero_dim width \c_zero_dim
2977     }
2978     c
2979     < {
2980       \vrule height \c_zero_dim depth \box_dp:N \@@arstrutbox width \c_zero_dim
2981       \end { minipage }
2982       \@@_cell_end:
2983     }
2984   }

```

We test for the presence of a <.

```

2985   \@@_make_m_preamble_x:n
2986 }

```

For w and W

```

2987 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2988 {
2989   \tl_gput_right:Nn \g_@@_preamble_tl
2990   {
2991     > {
2992       \dim_set:Nn \l_@@_col_width_dim { #4 }
2993       \hbox_set:Nw \l_@@_cell_box
2994       \@@_cell_begin:
2995       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2996     }
2997     c
2998     < {
2999       \@@_cell_end:
3000       \hbox_set_end:
3001       \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3002       #1
3003       \@@_adjust_size_box:
3004       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3005     }
3006   }

```

We test for the presence of a <.

```
3007 \@@_make_m_preamble_x:n
3008 }
```

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

```
3009 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3010 {
3011   \str_if_eq:nnTF { #1 } { < }
3012     { \@@_make_m_preamble_ix:n }
3013     { \@@_make_m_preamble:n { #1 } }
3014 }
3015 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3016 {
3017   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3018   \@@_make_m_preamble_x:n
3019 }
```

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`).

```
3020 \cs_new_protected:Npn \@@_put_box_in_flow:
3021 {
3022   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3023   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3024   \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
3025     { \box_use_drop:N \l_tmpa_box }
3026     { \@@_put_box_in_flow_i: }
3027 }
```

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

```
3028 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3029 {
3030   \pgfpicture
3031     \@@_qpoint:n { row - 1 }
3032     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3033     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3034     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3035     \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).

```
3036   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3037   {
3038     \int_set:Nn \l_tmpa_int
3039       {
3040         \str_range:Nnn
3041           \l_@@_baseline_tl
3042           6
3043           { \tl_count:o \l_@@_baseline_tl }
3044       }
3045     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3046   }
3047   {
3048     \str_if_eq:eeTF { \l_@@_baseline_tl } { t }
3049     { \int_set_eq:NN \l_tmpa_int \c_one_int }
3050     {
3051       \str_if_eq:onTF \l_@@_baseline_tl { b }
3052       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3053       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3054     }
3055   }
```



```

3054     }
3055     \bool_lazy_or:nnT
3056     { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3057     { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3058     {
3059         \@@_error:n { bad-value-for-baseline }
3060         \int_set_eq:NN \l_tmpa_int \c_one_int
3061     }
3062     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3063     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3064     }
3065     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3066     \endpgfpicture
3067     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3068     \box_use_drop:N \l_tmpa_box
3069 }

```

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).

```

3070 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3071 {

```

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.

```

3072     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3073     {
3074         \int_compare:nNnT { \c@jCol } > { \c_one_int }
3075         {
3076             \box_set_wd:Nn \l_@@_the_array_box
3077             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3078         }
3079     }

```

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

```

3080     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3081     \bool_if:NT \l_@@_caption_above_bool
3082     {
3083         \tl_if_empty:NF \l_@@_caption_tl
3084         {
3085             \bool_set_false:N \g_@@_caption_finished_bool
3086             \int_gzero:N \c@tabularnote
3087             \@@_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).

```

3088         \int_compare:nNnT { \g_@@_notes_caption_int } > { \c_zero_int }
3089         {
3090             \tl_gput_right:Ne \g_@@_aux_tl
3091             {
3092                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3093                 { \int_use:N \g_@@_notes_caption_int }
3094             }
3095             \int_gzero:N \g_@@_notes_caption_int
3096         }
3097     }
3098 }

```

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

```

3099   \hbox
3100   {
3101     \box_use_drop:N \l_@@_the_array_box

```

We have to draw the blocks right now 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.

```

3102     \@@_create_extra_nodes:
3103     \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3104   }

```

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).

```

3105   \bool_lazy_any:nT
3106   {
3107     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3108     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3109     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3110   }
3111   \@@_insert_tabularnotes:
3112   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3113   \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3114   \end { minipage }
3115 }

```

```

3116 \cs_new_protected:Npn \@@_insert_caption:
3117 {
3118   \tl_if_empty:NF \l_@@_caption_tl
3119   {
3120     \cs_if_exist:NTF \c@type
3121     { \@@_insert_caption_i: }
3122     { \@@_error:n { caption~outside~float } }
3123   }
3124 }

```

```

3125 \cs_new_protected:Npn \@@_insert_caption_i:
3126 {
3127   \group_begin:

```

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

```

3128   \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.

```

3129   \IfPackageLoadedT { floatrow }
3130   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3131   \tl_if_empty:NTF \l_@@_short_caption_tl
3132   { \caption }
3133   { \caption [ \l_@@_short_caption_tl ] }
3134   { \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.

```

3135 \bool_if:NF \g_@@_caption_finished_bool
3136 {
3137   \bool_gset_true:N \g_@@_caption_finished_bool
3138   \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3139   \int_gzero:N \c@tabularnote
3140 }
3141 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3142 \group_end:
3143 }
3144 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3145 {
3146   \@@_error_or_warning:n { tabularnote-below-the-tabular }
3147   \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3148 }
3149 \cs_new_protected:Npn \@@_insert_tabularnotes:
3150 {
3151   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3152   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3153   \skip_vertical:N 0.65ex

```

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

```

3154 \group_begin:
3155 \l_@@_notes_code_before_tl
3156 \tl_if_empty:NF \g_@@_tabularnote_tl
3157 {
3158   \g_@@_tabularnote_tl \par
3159   \tl_gclear:N \g_@@_tabularnote_tl
3160 }

```

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.

```

3161 \int_compare:nNnT { \c@tabularnote } > { \c_zero_int }
3162 {
3163   \bool_if:NTF \l_@@_notes_para_bool
3164   {
3165     \begin { tabularnotes* }
3166     \seq_map_inline:Nn \g_@@_notes_seq
3167       { \@@_one_tabularnote:nn ##1 }
3168     \strut
3169     \end { tabularnotes* }

```

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

```

3170 \par
3171 }
3172 {
3173   \tabularnotes
3174   \seq_map_inline:Nn \g_@@_notes_seq
3175     { \@@_one_tabularnote:nn ##1 }
3176   \strut
3177   \endtabularnotes
3178 }
3179 }
3180 \unskip
3181 \group_end:
3182 \bool_if:NT \l_@@_notes_bottomrule_bool
3183 {
3184   \IfPackageLoadedTF { booktabs }
3185   {

```

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

```

3186 \skip_vertical:N \aboverulesep

```

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

```

3187         { \CT@arc@ \hrule height \heavyrulewidth }
3188     }
3189     { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3190 }
3191 \l_@@_notes_code_after_tl
3192 \seq_gclear:N \g_@@_notes_seq
3193 \seq_gclear:N \g_@@_notes_in_caption_seq
3194 \int_gzero:N \c@tabularnote
3195 }

```

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.

```

3196 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3197 {
3198     \tl_if_novalue:nTF { #1 }
3199     { \item }
3200     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3201 }

```

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.

```

3202 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3203 {
3204     \pgfpicture
3205     \@@_qpoint:n { row - 1 }
3206     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3207     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3208     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3209     \endpgfpicture
3210     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3211     \int_if_zero:nT { \l_@@_first_row_int }
3212     {
3213         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3214         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3215     }
3216     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3217 }

```

Now, the general case.

```

3218 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3219 {

```

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

```

3220     \str_if_eq:eeT { \l_@@_baseline_tl } { t }
3221     { \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`.

```

3222     \pgfpicture
3223     \@@_qpoint:n { row - 1 }
3224     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3225     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3226     {
3227         \int_set:Nn \l_tmpa_int
3228         {
3229             \str_range:Nnn
3230             \l_@@_baseline_tl
3231             { 6 }
3232             { \tl_count:o \l_@@_baseline_tl }

```

```

3233     }
3234     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3235   }
3236   {
3237     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3238     \bool_lazy_or:nnT
3239       { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3240       { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3241     {
3242       \@@_error:n { bad-value-for~baseline }
3243       \int_set:Nn \l_tmpa_int 1
3244     }
3245     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3246   }
3247   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3248   \endpgfpicture
3249   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3250   \int_if_zero:nT { \l_@@_first_row_int }
3251   {
3252     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3253     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3254   }
3255   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3256 }

```

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.

```

3257 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3258 {

```

We will compute the real width of both delimiters used.

```

3259   \dim_zero_new:N \l_@@_real_left_delim_dim
3260   \dim_zero_new:N \l_@@_real_right_delim_dim
3261   \hbox_set:Nn \l_tmpb_box
3262   {
3263     \m@th % added 2024/11/21
3264     \c_math_toggle_token
3265     \left #1
3266     \vcenter
3267     {
3268       \vbox_to_ht:nn
3269         { \box_ht_plus_dp:N \l_tmpa_box }
3270       { }
3271     }
3272     \right .
3273     \c_math_toggle_token
3274   }
3275   \dim_set:Nn \l_@@_real_left_delim_dim
3276   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3277   \hbox_set:Nn \l_tmpb_box
3278   {
3279     \m@th % added 2024/11/21
3280     \c_math_toggle_token
3281     \left .
3282     \vbox_to_ht:nn
3283       { \box_ht_plus_dp:N \l_tmpa_box }
3284       { }
3285     \right #2
3286     \c_math_toggle_token
3287   }
3288   \dim_set:Nn \l_@@_real_right_delim_dim
3289   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3290     \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3291     \@@_put_box_in_flow:
3292     \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3293 }

```

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).

```

3294 \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).

```

3295 {
3296   \peek_remove_spaces:n
3297   {
3298     \peek_meaning:NTF \end
3299     { \@@_analyze_end:Nn }
3300     {
3301       \@@_transform_preamble:

```

Here is the call to `\array` (we have a dedicated macro `\@@_array:n` because of compatibility with the classes `revtex4-1` and `revtex4-2`).

```

3302       \@@_array:o \g_@@_array_preamble_tl
3303     }
3304   }
3305 }
3306 {
3307   \@@_create_col_nodes:
3308   \endarray
3309 }

```

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

```

3310 \NewDocumentEnvironment { @@-light-syntax } { b }
3311 {

```

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`.

```

3312   \tl_if_empty:nT { #1 }
3313   { \@@_fatal:n { empty-environment } }
3314   \tl_if_in:nnT { #1 } { & }
3315   { \@@_fatal:n { ampersand-in-light-syntax } }
3316   \tl_if_in:nnT { #1 } { \ }
3317   { \@@_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`.

```

3318   \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3319 }

```

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.

```

3320 {
3321   \@@_create_col_nodes:
3322   \endarray
3323 }

3324 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3325 {
3326   \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).

```

3327   \seq_clear_new:N \l_@@_rows_seq

```

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

```

3328   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3329   \bool_if:NTF \l_@@_light_syntax_expanded_bool
3330     { \seq_set_split:Nee }
3331     { \seq_set_split:Non }
3332   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3333   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3334   \tl_if_empty:NF \l_tmpa_tl
3335   { \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.

```

3336   \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
3337     { \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`).

```

3338   \tl_build_begin:N \l_@@_new_body_tl
3339   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3340   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3341   \@@_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).

```

3342   \seq_map_inline:Nn \l_@@_rows_seq
3343   {
3344     \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3345     \@@_line_with_light_syntax:n { ##1 }
3346   }
3347   \tl_build_end:N \l_@@_new_body_tl

3348   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
3349   {
3350     \int_set:Nn \l_@@_last_col_int
3351       { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3352   }

```

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.

```

3353   \@@_transform_preamble:

```

The call to `\array` is in the following command (we have a dedicated macro `\@@_array:` because of compatibility with the classes `revtex4-1` and `revtex4-2`).

```

3354   \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3355 }

```

```

3356 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3357 {
3358   \seq_clear_new:N \l_@@_cells_seq
3359   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3360   \int_set:Nn \l_@@_nb_cols_int
3361   {
3362     \int_max:nn
3363     { \l_@@_nb_cols_int }
3364     { \seq_count:N \l_@@_cells_seq }
3365   }
3366   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3367   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3368   \seq_map_inline:Nn \l_@@_cells_seq
3369   { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3370 }
3371 \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`.

```

3372 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3373 {
3374   \str_if_eq:eeT { \g_@@_name_env_str } { #2 }
3375   { \@@_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.

```

3376   \end { #2 }
3377 }

```

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`).

```

3378 \cs_new:Npn \@@_create_col_nodes:
3379 {
3380   \crrc
3381   \int_if_zero:nT { \l_@@_first_col_int }
3382   {
3383     \omit
3384     \hbox_overlap_left:n
3385     {
3386       \bool_if:NT \l_@@_code_before_bool
3387       { \pgfsys@markposition { \@@_env: - col - 0 } }
3388       \pgfpicture
3389       \pgfrememberpicturepositiononpagetrue
3390       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3391       \str_if_empty:NF \l_@@_name_str
3392       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3393       \endpgfpicture
3394       \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3395     }
3396     &
3397   }
3398   \omit

```

The following instruction must be put after the instruction `\omit`.

```

3399   \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`).

```

3400   \int_if_zero:nTF { \l_@@_first_col_int }
3401   {
3402     \@@_mark_position:n { 1 }

```



```

3403 \pgfpicture
3404 \pgfrememberpicturepositiononpagetrue
3405 \pgfcoordinate { \@@_env: - col - 1 }
3406 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3407 \str_if_empty:NF \l_@@_name_str
3408 { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3409 \endpgfpicture
3410 }
3411 {
3412 \bool_if:NT \l_@@_code_before_bool
3413 {
3414 \hbox
3415 {
3416 \skip_horizontal:n { 0.5 \arrayrulewidth }
3417 \pgfsys@markposition { \@@_env: - col - 1 }
3418 \skip_horizontal:n { -0.5 \arrayrulewidth }
3419 }
3420 }
3421 \pgfpicture
3422 \pgfrememberpicturepositiononpagetrue
3423 \pgfcoordinate { \@@_env: - col - 1 }
3424 { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3425 \@@_node_alias:n { 1 }
3426 \endpgfpicture
3427 }

```

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.

```

3428 \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
3429 \bool_if:NF \l_@@_auto_columns_width_bool
3430 { \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim } }
3431 {
3432 \bool_lazy_and:nnTF
3433 { \l_@@_auto_columns_width_bool }
3434 { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3435 { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3436 { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3437 \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3438 }
3439 \skip_horizontal:N \g_tmpa_skip
3440 \hbox
3441 {
3442 \@@_mark_position:n { 2 }
3443 \pgfpicture
3444 \pgfrememberpicturepositiononpagetrue
3445 \pgfcoordinate { \@@_env: - col - 2 }
3446 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3447 \@@_node_alias:n { 2 }
3448 \endpgfpicture
3449 }

```

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.

```

3450 \int_gset_eq:NN \g_tmpa_int \c_one_int
3451 \bool_if:NTF \g_@@_last_col_found_bool
3452 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3453 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3454 {
3455 &
3456 \omit

```

```
3457 \int_gincr:N \g_tmpa_int
```

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

```
3458 \skip_horizontal:N \g_tmpa_skip
3459 \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }
```

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

```
3460 \pgfpicture
3461 \pgfrememberpicturepositiononpagetrue
3462 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3463 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3464 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3465 \endpgfpicture
3466 }
```

```
3467 &
3468 \omit
```

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

```
3469 \bool_lazy_or:nnF
3470 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3471 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3472 {
3473   \skip_horizontal:N \g_tmpa_skip
3474   \int_gincr:N \g_tmpa_int
3475   \bool_lazy_any:nF
3476   {
3477     \g_@@_delims_bool
3478     \l_@@_tabular_bool
3479     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3480     \l_@@_exterior_arraycolsep_bool
3481     \l_@@_bar_at_end_of_pream_bool
3482   }
3483   { \skip_horizontal:n { - \col@sep } }
3484   \bool_if:NT \l_@@_code_before_bool
3485   {
3486     \hbox
3487     {
3488       \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.

```
3489 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3490 { \skip_horizontal:n { - \arraycolsep } }
3491 \pgfsys@markposition
3492 { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3493 \skip_horizontal:n { 0.5 \arrayrulewidth }
3494 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3495 { \skip_horizontal:N \arraycolsep }
3496 }
3497 }
3498 \pgfpicture
3499 \pgfrememberpicturepositiononpagetrue
3500 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3501 {
3502   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3503   {
3504     \pgfpoint
3505     { - 0.5 \arrayrulewidth - \arraycolsep }
3506     \c_zero_dim
3507   }
3508   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }

```

```

3509         }
3510         \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3511     \endpgfpicture
3512 }

3513 \bool_if:NT \g_@@_last_col_found_bool
3514 {
3515     \hbox_overlap_right:n
3516     {
3517         \skip_horizontal:N \g_@@_width_last_col_dim
3518         \skip_horizontal:N \col@sep
3519         \bool_if:NT \l_@@_code_before_bool
3520         {
3521             \pgfsys@markposition
3522             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3523         }
3524         \pgfpicture
3525         \pgfrememberpicturepositiononpagetrue
3526         \pgfcoordinate
3527         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3528         \pgfpointorigin
3529         \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3530     \endpgfpicture
3531 }
3532 }
3533 % \cr
3534 }

3535 \cs_new_protected:Npn \@@_mark_position:n #1
3536 {
3537     \bool_if:NT \l_@@_code_before_bool
3538     {
3539         \hbox
3540         {
3541             \skip_horizontal:n { -0.5 \arrayrulewidth }
3542             \pgfsys@markposition { \@@_env: - col - #1 }
3543             \skip_horizontal:n { 0.5 \arrayrulewidth }
3544         }
3545     }
3546 }

3547 \cs_new_protected:Npn \@@_node_alias:n #1
3548 {
3549     \str_if_empty:NF \l_@@_name_str
3550     { \pgfnodealias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3551 }

```

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

```

3552 \tl_const:Nn \c_@@_preamble_first_col_tl
3553 {
3554     >
3555     {

```

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

```

3556     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3557     \bool_gset_true:N \g_@@_after_col_zero_bool
3558     \@@_begin_of_row:
3559     \hbox_set:Nw \l_@@_cell_box
3560     \@@_math_toggle:
3561     \@@_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”.

```

3562     \int_compare:nNt { \c@iRow } > { \c_zero_int }
3563     {
3564         \bool_lazy_or:nnT
3565         { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3566         { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3567         {
3568             \l_@@_code_for_first_col_tl
3569             \xglobal \colorlet { nicematrix-first-col } { . }
3570         }
3571     }
3572 }

```

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`.

```

3573     l
3574     <
3575     {
3576         \@@_math_toggle:
3577         \hbox_set_end:
3578         \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3579         \@@_adjust_size_box:
3580         \@@_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.

```

3581     \dim_gset:Nn \g_@@_width_first_col_dim
3582     { \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.

```

3583     \hbox_overlap_left:n
3584     {
3585         \dim_compare:nNtF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3586         { \@@_node_cell: }
3587         { \box_use_drop:N \l_@@_cell_box }
3588         \skip_horizontal:N \l_@@_left_delim_dim
3589         \skip_horizontal:N \l_@@_left_margin_dim
3590         \skip_horizontal:N \l_@@_extra_left_margin_dim
3591     }
3592     \bool_gset_false:N \g_@@_empty_cell_bool
3593     \skip_horizontal:n { -2 \col@sep }
3594 }
3595 }

```

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

```

3596 \tl_const:Nn \c_@@_preamble_last_col_tl
3597 {
3598     >
3599     {
3600         \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).

```

3601     \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.

```

3602     \bool_gset_true:N \g_@@_last_col_found_bool
3603     \int_gincr:N \c@jCol
3604     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3605     \hbox_set:Nw \l_@@_cell_box
3606     \@@_math_toggle:
3607     \@@_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”.

```

3608     \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3609     {
3610         \bool_lazy_or:nnT
3611         { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3612         { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3613         {
3614             \l_@@_code_for_last_col_tl
3615             \xglobal \colorlet { nicematrix-last-col } { . }
3616         }
3617     }
3618 }
3619 1
3620 <
3621 {
3622     \@@_math_toggle:
3623     \hbox_set_end:
3624     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3625     \@@_adjust_size_box:
3626     \@@_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.

```

3627     \dim_gset:Nn \g_@@_width_last_col_dim
3628     { \dim_max:nn { \g_@@_width_last_col_dim } { \box_wd:N \l_@@_cell_box } }
3629     \skip_horizontal:n { -2 \col@sep }

```

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

```

3630     \hbox_overlap_right:n
3631     {
3632         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3633         {
3634             \skip_horizontal:N \l_@@_right_delim_dim
3635             \skip_horizontal:N \l_@@_right_margin_dim
3636             \skip_horizontal:N \l_@@_extra_right_margin_dim
3637             \@@_node_cell:
3638         }
3639     }
3640     \bool_gset_false:N \g_@@_empty_cell_bool
3641 }
3642 }

```

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

```

3643 \NewDocumentEnvironment { NiceArray } { }
3644 {
3645     \bool_gset_false:N \g_@@_delims_bool
3646     \str_if_empty:NT \g_@@_name_env_str
3647     { \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).

```

3648     \NiceArrayWithDelims . .
3649 }
3650 { \endNiceArrayWithDelims }

```

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

```

3651 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3652 {
3653     \NewDocumentEnvironment { #1 NiceArray } { }
3654     {

```

```

3655 \bool_gset_true:N \g_@@_delims_bool
3656 \str_if_empty:NT \g_@@_name_env_str
3657 { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3658 \@@_test_if_math_mode:
3659 \NiceArrayWithDelims #2 #3
3660 }
3661 { \endNiceArrayWithDelims }
3662 }
3663 \@@_def_env:NNN p ( )
3664 \@@_def_env:NNN b [ ]
3665 \@@_def_env:NNN B \{ \}
3666 \@@_def_env:NNN v \vert \vert
3667 \@@_def_env:NNN V \Vert \Vert

```

13 The environment `{NiceMatrix}` and its variants

```

3668 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3669 {
3670 \bool_set_false:N \l_@@_preamble_bool
3671 \tl_clear:N \l_tmpa_tl
3672 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3673 { \tl_set:Nn \l_tmpa_tl { @ { } } }
3674 \tl_put_right:Nn \l_tmpa_tl
3675 {
3676 *
3677 {
3678 \int_case:nnF \l_@@_last_col_int
3679 {
3680 { -2 } { \c@MaxMatrixCols }
3681 { -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).

```

3682 }
3683 { \int_eval:n { \l_@@_last_col_int - 1 } }
3684 }
3685 { #2 }
3686 }
3687 \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3688 \exp_args:No \l_tmpb_tl \l_tmpa_tl
3689 }
3690 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3691 \clist_map_inline:nn { p , b , B , v , V }
3692 {
3693 \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3694 {
3695 \bool_gset_true:N \g_@@_delims_bool
3696 \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3697 \int_if_zero:nT { \l_@@_last_col_int }
3698 {
3699 \bool_set_true:N \l_@@_last_col_without_value_bool
3700 \int_set:Nn \l_@@_last_col_int { -1 }
3701 }
3702 \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3703 \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3704 }
3705 { \use:c { end #1 NiceArray } }
3706 }

```

We define also an environment `{NiceMatrix}`

```

3707 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3708 {
3709   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3710   \int_if_zero:nT { \l_@@_last_col_int }
3711   {
3712     \bool_set_true:N \l_@@_last_col_without_value_bool
3713     \int_set:Nn \l_@@_last_col_int { -1 }
3714   }
3715   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3716   \bool_lazy_or:nnT
3717     { \clist_if_empty_p:N \l_@@_vlines_clist }
3718     { \l_@@_except_borders_bool }
3719     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3720   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3721 }
3722 { \endNiceArray }

```

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

```

3723 \cs_new_protected:Npn \@@_NotEmpty:
3724 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3725 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3726 {

```

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.

```

3727   \dim_compare:nNnT { \l_@@_width_dim } = { \c_zero_dim }
3728     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3729   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3730   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3731   \tl_if_empty:NF \l_@@_short_caption_tl
3732   {
3733     \tl_if_empty:NT \l_@@_caption_tl
3734     {
3735       \@@_error_or_warning:n { short-caption-without-caption }
3736       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3737     }
3738   }
3739   \tl_if_empty:NF \l_@@_label_tl
3740   {
3741     \tl_if_empty:NT \l_@@_caption_tl
3742     { \@@_error_or_warning:n { label-without-caption } }
3743   }
3744   \NewDocumentEnvironment { TabularNote } { b }
3745   {
3746     \bool_if:NTF \l_@@_in_code_after_bool
3747       { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3748       {
3749         \tl_if_empty:NF \g_@@_tabularnote_tl
3750         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3751         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3752       }
3753   }
3754   { }
3755   \@@_settings_for_tabular:
3756   \NiceArray { #2 }
3757 }
3758 { \endNiceArray }
3759 \cs_new_protected:Npn \@@_settings_for_tabular:
3760 {

```

```

3761 \bool_set_true:N \l_@@_tabular_bool
3762 \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3763 \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3764 \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3765 }

3766 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3767 {
3768   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3769   \dim_set:Nn \l_@@_width_dim { #1 }
3770   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3771   \@@_settings_for_tabular:
3772   \NiceArray { #3 }
3773 }
3774 {
3775   \endNiceArray
3776   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3777     { \@@_error:n { NiceTabularX~without~X } }
3778 }

3779 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3780 {
3781   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3782   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3783   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3784   \@@_settings_for_tabular:
3785   \NiceArray { #3 }
3786 }
3787 { \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`).

```

3788 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3789 {
3790   \bool_lazy_all:nT
3791   {
3792     { \dim_compare_p:nNn { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim } }
3793     { \l_@@_hvlines_bool }
3794     { ! \g_@@_delims_bool }
3795     { ! \l_@@_except_borders_bool }
3796   }
3797   {
3798     \bool_set_true:N \l_@@_except_borders_bool
3799     \clist_if_empty:NF \l_@@_corners_clist
3800     { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3801     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3802     {
3803       \@@_stroke_block:nnn
3804       {
3805         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3806         draw = \l_@@_rules_color_tl
3807       }
3808       { 1-1 }
3809       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3810     }
3811   }
3812 }

```



```

3813 \cs_new_protected:Npn \@@_after_array:
3814 {

```

There was a `\hook_gput_code:nnn { env / tabular / begin } { nicematrix }` in the command `\@@_pre_array_ii:` 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`.

```

3815 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3816 \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.

```

3817 \bool_if:NT \g_@@_last_col_found_bool
3818 { \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`.

```

3819 \bool_if:NT \l_@@_last_col_without_value_bool
3820 { \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.

```

3821 \bool_if:NT \l_@@_last_row_without_value_bool
3822 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3823 \tl_gput_right:Ne \g_@@_aux_tl
3824 {
3825   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3826   {
3827     \int_use:N \l_@@_first_row_int ,
3828     \int_use:N \c@iRow ,
3829     \int_use:N \g_@@_row_total_int ,
3830     \int_use:N \l_@@_first_col_int ,
3831     \int_use:N \c@jCol ,
3832     \int_use:N \g_@@_col_total_int
3833   }
3834 }

```

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`).

```

3835 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3836 {
3837   \tl_gput_right:Ne \g_@@_aux_tl
3838   {
3839     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3840     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3841   }
3842 }
3843 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3844 {
3845   \tl_gput_right:Ne \g_@@_aux_tl
3846   {
3847     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3848     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3849     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3850     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3851   }
3852 }

```

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

```
3853 \@@_create_diag_nodes:
```

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

```
3854 \pgfpicture
3855 \@@_create_aliases_last:
3856 \str_if_empty:NF \l_@@_name_str { \@@_create_alias_nodes: }
3857 \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.

```
3858 \bool_if:NT \l_@@_parallelize_diags_bool
3859 {
3860   \int_gzero:N \g_@@_ddots_int
3861   \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.

```
3862   \dim_gzero:N \g_@@_delta_x_one_dim
3863   \dim_gzero:N \g_@@_delta_y_one_dim
3864   \dim_gzero:N \g_@@_delta_x_two_dim
3865   \dim_gzero:N \g_@@_delta_y_two_dim
3866 }
3867 \bool_set_false:N \l_@@_initial_open_bool
3868 \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.

```
3869 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }
```

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

```
3870 \@@_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.

```
3871 \clist_if_empty:NF \l_@@_corners_clist
3872 {
3873   \bool_if:NTF \l_@@_no_cell_nodes_bool
3874   { \@@_error:n { corners~with~no~cell~nodes } }
3875   { \@@_compute_corners: }
3876 }
```

The sequence `\g_@@_pos_of_blocks_seq` must be “adjusted” (for the case where the user have written something like `\Block{1-*}`).

```
3877 \@@_adjust_pos_of_blocks_seq:
3878 \@@_deal_with_rounded_corners:
3879 \clist_if_empty:NF \l_@@_hlines_clist { \@@_draw_hlines: }
3880 \clist_if_empty:NF \l_@@_vlines_clist { \@@_draw_vlines: }
```

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

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

```

3881 \IfPackageLoadedT { tikz }
3882 {
3883   \tikzset
3884   {
3885     every-picture / .style =
3886     {
3887       overlay ,
3888       remember-picture ,
3889       name-prefix = \@@_env: -
3890     }
3891   }
3892 }
3893 \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
3894 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3895 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3896 \cs_set_eq:NN \OverBrace \@@_OverBrace
3897 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3898 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3899 \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.

```

3900 \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
3901 \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.

```

3902 \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).

```

3903 \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`).

```

3904 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3905 { \@@_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:`.

```

3906 \bool_set_true:N \l_@@_in_code_after_bool
3907 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3908 \scan_stop:
3909 \tl_gclear:N \g_nicematrix_code_after_tl
3910 \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.

```

3911 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3912 \tl_if_empty:NF \g_@@_pre_code_before_tl
3913 {
3914   \tl_gput_right:Ne \g_@@_aux_tl
3915   {
3916     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3917     { \exp_not:o \g_@@_pre_code_before_tl }
3918   }
3919   \tl_gclear:N \g_@@_pre_code_before_tl
3920 }

```

```

3921 \tl_if_empty:NF \g_nicematrix_code_before_tl
3922 {
3923   \tl_gput_right:Ne \g_@@_aux_tl
3924   {
3925     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3926     { \exp_not:o \g_nicematrix_code_before_tl }
3927   }
3928   \tl_gclear:N \g_nicematrix_code_before_tl
3929 }

3930 \str_gclear:N \g_@@_name_env_str
3931 \@@_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`.

```

3932 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3933 }

3934 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
3935 {
3936   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3937   \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }

```

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

```

3938   \dim_set:Nn \l_@@_xdots_shorten_start_dim
3939   { 0.6 \l_@@_xdots_shorten_start_dim }
3940   \dim_set:Nn \l_@@_xdots_shorten_end_dim
3941   { 0.6 \l_@@_xdots_shorten_end_dim }
3942 }

```

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`.

```

3943 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3944 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

3945 \cs_new_protected:Npn \@@_create_alias_nodes:
3946 {
3947   \int_step_inline:nn { \c@iRow }
3948   {
3949     \pgfnodealias
3950     { \l_@@_name_str - ##1 - last }
3951     { \@@_env: - ##1 - \int_use:N \c@jCol }
3952   }
3953   \int_step_inline:nn { \c@jCol }
3954   {
3955     \pgfnodealias
3956     { \l_@@_name_str - last - ##1 }
3957     { \@@_env: - \int_use:N \c@iRow - ##1 }
3958   }
3959   \pgfnodealias
3960   { \l_@@_name_str - last - last }
3961   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
3962 }

```

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).

```

3963 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3964 {
3965   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3966   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3967 }

```

The following command must *not* be protected.

```

3968 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3969 {
3970   { #1 }
3971   { #2 }
3972   {
3973     \int_compare:nNnTF { #3 } > { 98 }
3974     { \int_use:N \c@iRow }
3975     { #3 }
3976   }
3977   {
3978     \int_compare:nNnTF { #4 } > { 98 }
3979     { \int_use:N \c@jCol }
3980     { #4 }
3981   }
3982   { #5 }
3983 }

```

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).

```

3984 \hook_gput_code:nnn { begindocument } { . }
3985 {
3986   \cs_new_protected:Npe \@@_draw_dotted_lines:
3987   {
3988     \c_@@_pgfortikzpicture_tl
3989     \@@_draw_dotted_lines_i:
3990     \c_@@_endpgfortikzpicture_tl
3991   }
3992 }

```

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

```

3993 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3994 {
3995   \pgfrememberpicturepositiononpagetrue
3996   \pgf@relevantforpicturesizefalse
3997   \g_@@_HVdotsfor_lines_tl
3998   \g_@@_Vdots_lines_tl
3999   \g_@@_Ddots_lines_tl
4000   \g_@@_Idots_lines_tl
4001   \g_@@_Cdots_lines_tl
4002   \g_@@_Ldots_lines_tl
4003 }

4004 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4005 {
4006   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4007   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4008 }

```

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

```

4009 \pgfdeclareshape { @@_diag_node }
4010 {
4011   \savedanchor { \five }
4012   {
4013     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4014     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4015   }
4016   \anchor { 5 } { \five }
4017   \anchor { center } { \pgfpointorigin }
4018   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4019   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4020   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4021   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4022   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4023   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4024   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4025   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4026   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4027   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4028 }

```

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).

```

4029 \cs_new_protected:Npn \@@_create_diag_nodes:
4030 {
4031   \pgfpicture
4032   \pgfrememberpicturepositiononpagetrue
4033   \int_step_inline:nn { \int_max:nn { \c@iRow } { \c@jCol } }
4034   {
4035     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4036     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4037     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4038     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4039     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4040     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4041     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4042     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4043     \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.

```

4044     \dim_set:Nn \l_tmpa_int { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4045     \dim_set:Nn \l_tmpb_int { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4046     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4047     \str_if_empty:NF \l_@@_name_str
4048     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4049   }

```

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

```

4050     \int_set:Nn \l_tmpa_int { \int_max:nn { \c@iRow } { \c@jCol } + 1 }
4051     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4052     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4053     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4054     \pgfcoordinate
4055     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4056     \pgfnodealias
4057     { \@@_env: - last }
4058     { \@@_env: - \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
4059     \str_if_empty:NF \l_@@_name_str
4060     {

```

```

4061 \pgfnodealias
4062 { \l_@@_name_str - \int_use:N \l_tmpa_int }
4063 { \@@_env: - \int_use:N \l_tmpa_int }
4064 \pgfnodealias
4065 { \l_@@_name_str - last }
4066 { \@@_env: - last }
4067 }
4068 \endpgfpicture
4069 }

```

16 We draw the dotted lines

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

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

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

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

This command computes:

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

```

4070 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4071 {

```

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

```

4072 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4073 \int_set:Nn \l_@@_initial_i_int { #1 }
4074 \int_set:Nn \l_@@_initial_j_int { #2 }
4075 \int_set:Nn \l_@@_final_i_int { #1 }
4076 \int_set:Nn \l_@@_final_j_int { #2 }

```

We will do two loops: one when determining the initial cell and the other when determining the final cell. The boolean `\l_@@_stop_loop_bool` will be used to control these loops. In the first loop, we search the “final” extremity of the line.

```

4077 \bool_set_false:N \l_@@_stop_loop_bool
4078 \bool_do_until:Nn \l_@@_stop_loop_bool
4079 {
4080 \int_add:Nn \l_@@_final_i_int { #3 }
4081 \int_add:Nn \l_@@_final_j_int { #4 }
4082 \bool_set_false:N \l_@@_final_open_bool

```

We test if we are still in the matrix. Since this is the core of the loop, we **optimize** the code by using a TeX-style of conditionals.

```

4083     \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4084     \if_int_compare:w #3 = \c_one_int
4085     \bool_set_true:N \l_@@_final_open_bool
4086   \else:
4087     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4088     \bool_set_true:N \l_@@_final_open_bool
4089   \fi:
4090 \fi:
4091 \else:
4092   \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4093   \if_int_compare:w #4 = -1
4094     \bool_set_true:N \l_@@_final_open_bool
4095   \fi:
4096 \else:
4097   \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4098   \if_int_compare:w #4 = \c_one_int
4099     \bool_set_true:N \l_@@_final_open_bool
4100   \fi:
4101 \fi:
4102 \fi:
4103 \fi:
4104 \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.

```

4105   {

```

We do a step backwards.

```

4106     \int_sub:Nn \l_@@_final_i_int { #3 }
4107     \int_sub:Nn \l_@@_final_j_int { #4 }
4108     \bool_set_true:N \l_@@_stop_loop_bool
4109   }

```

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`.

```

4110   {
4111     \cs_if_exist:cTF
4112     {
4113       @@ _ dotted _
4114       \int_use:N \l_@@_final_i_int -
4115       \int_use:N \l_@@_final_j_int
4116     }
4117     {
4118       \int_sub:Nn \l_@@_final_i_int { #3 }
4119       \int_sub:Nn \l_@@_final_j_int { #4 }
4120       \bool_set_true:N \l_@@_final_open_bool
4121       \bool_set_true:N \l_@@_stop_loop_bool
4122     }
4123   }
4124   \cs_if_exist:cTF
4125   {
4126     pgf @ sh @ ns @ \@@_env:
4127     - \int_use:N \l_@@_final_i_int
4128     - \int_use:N \l_@@_final_j_int
4129   }
4130   { \bool_set_true:N \l_@@_stop_loop_bool }

```

If the case is empty, we declare that the cell as non-empty. Indeed, we will draw a dotted line and the cell will be on that dotted line. All the cells of a dotted line have to be marked as “dotted” because we don't want intersections between dotted lines. We recall that the research of the extremities of the lines are all done in the same TeX group (the group of the environment), even though, when the extremities are found, each line is drawn in a TeX group that we will open for the options of the line.

```

4131   {

```



```

4132         \cs_set_nopar:cpn
4133         {
4134             @@ _ dotted _
4135             \int_use:N \l_@@_final_i_int -
4136             \int_use:N \l_@@_final_j_int
4137         }
4138         { }
4139     }
4140 }
4141 }
4142 }

```

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

```

4143     \bool_set_false:N \l_@@_stop_loop_bool

```

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

```

4144     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4145     \bool_do_until:Nn \l_@@_stop_loop_bool
4146     {
4147         \int_sub:Nn \l_@@_initial_i_int { #3 }
4148         \int_sub:Nn \l_@@_initial_j_int { #4 }
4149         \bool_set_false:N \l_@@_initial_open_bool

```

We test if we are still in the matrix. Since this is the core of the loop, we **optimize** the code by using a TeX-style of conditionals.

```

4150     \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4151     \if_int_compare:w #3 = \c_one_int
4152     \bool_set_true:N \l_@@_initial_open_bool
4153     \else:

```

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

```

4154     \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4155     \bool_set_true:N \l_@@_initial_open_bool
4156     \fi:
4157     \fi:
4158     \else:
4159     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4160     \if_int_compare:w #4 = \c_one_int
4161     \bool_set_true:N \l_@@_initial_open_bool
4162     \fi:
4163     \else:
4164     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4165     \if_int_compare:w #4 = -1
4166     \bool_set_true:N \l_@@_initial_open_bool
4167     \fi:
4168     \fi:
4169     \fi:
4170     \fi:
4171     \bool_if:NTF \l_@@_initial_open_bool
4172     {
4173         \int_add:Nn \l_@@_initial_i_int { #3 }
4174         \int_add:Nn \l_@@_initial_j_int { #4 }
4175         \bool_set_true:N \l_@@_stop_loop_bool
4176     }
4177     {
4178         \cs_if_exist:cTF
4179         {
4180             @@ _ dotted _
4181             \int_use:N \l_@@_initial_i_int -
4182             \int_use:N \l_@@_initial_j_int
4183         }

```

```

4184     {
4185         \int_add:Nn \l_@@_initial_i_int { #3 }
4186         \int_add:Nn \l_@@_initial_j_int { #4 }
4187         \bool_set_true:N \l_@@_initial_open_bool
4188         \bool_set_true:N \l_@@_stop_loop_bool
4189     }
4190     {
4191         \cs_if_exist:cTF
4192         {
4193             pgf @ sh @ ns @ \@@_env:
4194             - \int_use:N \l_@@_initial_i_int
4195             - \int_use:N \l_@@_initial_j_int
4196         }
4197         { \bool_set_true:N \l_@@_stop_loop_bool }
4198         {
4199             \cs_set_nopar:cpn
4200             {
4201                 @@ _ dotted _
4202                 \int_use:N \l_@@_initial_i_int -
4203                 \int_use:N \l_@@_initial_j_int
4204             }
4205             { }
4206         }
4207     }
4208 }
4209 }

```

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.

```

4210 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4211 {
4212     { \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`.

```

4213     { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4214     { \int_use:N \l_@@_final_i_int }
4215     { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4216     { } % for the name of the block
4217 }
4218 }

```

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 analyse 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.

```

4219 \cs_new_protected:Npn \@@_open_shorten:
4220 {
4221     \bool_if:NT \l_@@_initial_open_bool
4222     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4223     \bool_if:NT \l_@@_final_open_bool
4224     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4225 }

```

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).

```

4226 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4227 {
4228     \int_set_eq:NN \l_@@_row_min_int \c_one_int

```

```

4229 \int_set_eq:NN \l_@@_col_min_int \c_one_int
4230 \int_set_eq:NN \l_@@_row_max_int \c@iRow
4231 \int_set_eq:NN \l_@@_col_max_int \c@jCol

```

We do a loop over all the submatrices specified in the `code-before`. We have stored the position of all those submatrices in `\g_@@_submatrix_seq`.

```

4232 \seq_if_empty:NF \g_@@_submatrix_seq
4233 {
4234   \seq_map_inline:Nn \g_@@_submatrix_seq
4235   { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4236 }
4237 }

```

#1 and **#2** are the numbers of row and columns of the cell where the command of dotted line (ex.: `\Vdots`) has been issued. **#3**, **#4**, **#5** and **#6** are the specification (in *i* and *j*) of the submatrix we are analyzing.

Here is the programming of that command with the the standard syntax of L3.

```

\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
{
  \bool_if:nT
  {
    \int_compare_p:n { #3 <= #1 <= #5 }
    &&
    \int_compare_p:n { #4 <= #2 <= #6 }
  }
  {
    \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
    \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
    \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
    \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
  }
}

```

However, for efficiency, we will use the following version.

```

4238 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4239 {
4240   \if_int_compare:w #3 > #1
4241   \else:
4242     \if_int_compare:w #1 > #5
4243     \else:
4244       \if_int_compare:w #4 > #2
4245       \else:
4246         \if_int_compare:w #2 > #6
4247         \else:
4248           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4249           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4250           \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4251           \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4252         \fi:
4253       \fi:
4254     \fi:
4255   \fi:
4256 }

4257 \cs_new_protected:Npn \@@_set_initial_coords:
4258 {
4259   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4260   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4261 }
4262 \cs_new_protected:Npn \@@_set_final_coords:
4263 {

```

```

4264 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4265 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4266 }
4267 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4268 {
4269 \pgfpointanchor
4270 {
4271 \@@_env:
4272 - \int_use:N \l_@@_initial_i_int
4273 - \int_use:N \l_@@_initial_j_int
4274 }
4275 { #1 }
4276 \@@_set_initial_coords:
4277 }
4278 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4279 {
4280 \pgfpointanchor
4281 {
4282 \@@_env:
4283 - \int_use:N \l_@@_final_i_int
4284 - \int_use:N \l_@@_final_j_int
4285 }
4286 { #1 }
4287 \@@_set_final_coords:
4288 }
4289 \cs_new_protected:Npn \@@_open_x_initial_dim:
4290 {
4291 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4292 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4293 {
4294 \cs_if_exist:cT
4295 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4296 {
4297 \pgfpointanchor
4298 { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4299 { west }
4300 \dim_set:Nn \l_@@_x_initial_dim
4301 { \dim_min:nn { \l_@@_x_initial_dim } { \pgf@x } }
4302 }
4303 }

```

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

```

4304 \dim_compare:nNnT { \l_@@_x_initial_dim } = { \c_max_dim }
4305 {
4306 \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4307 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4308 \dim_add:Nn \l_@@_x_initial_dim \col@sep
4309 }
4310 }
4311 \cs_new_protected:Npn \@@_open_x_final_dim:
4312 {
4313 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4314 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4315 {
4316 \cs_if_exist:cT
4317 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4318 {
4319 \pgfpointanchor
4320 { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4321 { east }
4322 \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
4323 { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4324 }

```

```
4325 }
```

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

```
4326 \dim_compare:nNnT { \l_@@_x_final_dim } = { - \c_max_dim }
4327 {
4328   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4329   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4330   \dim_sub:Nn \l_@@_x_final_dim \col@sep
4331 }
4332 }
```

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.

```
4333 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4334 {
4335   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4336   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4337   {
4338     \@@_find_extremities_of_line:nnnn { #1 } { #2 } 0 1
```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```
4339   \group_begin:
4340   \@@_open_shorten:
4341   \int_if_zero:nTF { #1 }
4342   { \color { nicematrix-first-row } }
4343   {
```

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.

```
4344     \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4345     { \color { nicematrix-last-row } }
4346   }
4347   \keys_set:nn { nicematrix / xdots } { #3 }
4348   \@@_color:o \l_@@_xdots_color_tl
4349   \@@_actually_draw_Ldots:
4350   \group_end:
4351 }
4352 }
```

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`.

```
4353 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4354 {
4355   \bool_if:NTF \l_@@_initial_open_bool
4356   {
4357     \@@_open_x_initial_dim:
4358     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4359     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4360   }
4361   { \@@_set_initial_coords_from_anchor:n { base-east } }
```

```

4362 \bool_if:NTF \l_@@_final_open_bool
4363 {
4364   \@@_open_x_final_dim:
4365   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4366   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4367 }
4368 { \@@_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.

```

4369 \bool_lazy_all:nTF
4370 {
4371   \l_@@_initial_open_bool
4372   \l_@@_final_open_bool
4373   { \int_compare_p:nNn { \l_@@_initial_i_int } = { \l_@@_last_row_int } }
4374 }
4375 {
4376   \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4377   \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4378 }

```

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

```

4379 {
4380   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4381   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4382 }
4383 \@@_draw_line:
4384 }

```

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.

```

4385 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4386 {
4387   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4388   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4389   {
4390     \@@_find_extremities_of_line:nnnn { #1 } { #2 } { 0 } { 1 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4391 \group_begin:
4392 \@@_open_shorten:
4393 \int_if_zero:nTF { #1 }
4394 { \color { nicematrix-first-row } }
4395 {

```

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.

```

4396   \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4397   { \color { nicematrix-last-row } }
4398 }
4399 \keys_set:nn { nicematrix / xdots } { #3 }
4400 \@@_color:o \l_@@_xdots_color_tl
4401 \@@_actually_draw_Cdots:
4402 \group_end:
4403 }
4404 }

```

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.

```

4405 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4406 {
4407   \bool_if:NTF \l_@@_initial_open_bool
4408     { \@@_open_x_initial_dim: }
4409     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4410   \bool_if:NTF \l_@@_final_open_bool
4411     { \@@_open_x_final_dim: }
4412     { \@@_set_final_coords_from_anchor:n { mid-west } }
4413   \bool_lazy_and:nnTF
4414     { \l_@@_initial_open_bool }
4415     { \l_@@_final_open_bool }
4416   {
4417     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4418     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4419     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4420     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4421     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4422   }
4423   {
4424     \bool_if:NT \l_@@_initial_open_bool
4425       { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4426     \bool_if:NT \l_@@_final_open_bool
4427       { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4428   }
4429   \@@_draw_line:
4430 }
4431 \cs_new_protected:Npn \@@_open_y_initial_dim:
4432 {
4433   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4434   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4435   {
4436     \cs_if_exist:cT
4437       { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4438       {
4439         \pgfpointanchor
4440           { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4441           { north }
4442         \dim_compare:nNnT { \pgf@y } > { \l_@@_y_initial_dim }
4443           { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4444       }
4445   }
4446   \dim_compare:nNnT { \l_@@_y_initial_dim } = { - \c_max_dim }
4447   {
4448     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4449     \dim_set:Nn \l_@@_y_initial_dim
4450       {
4451         \fp_to_dim:n
4452           {
4453             \pgf@y
4454             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4455           }
4456       }
4457   }
4458 }

```

```

4459 \cs_new_protected:Npn \@@_open_y_final_dim:
4460 {
4461   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4462   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4463   {
4464     \cs_if_exist:cT
4465     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4466     {
4467       \pgfpointanchor
4468       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4469       { south }
4470       \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
4471       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4472     }
4473   }
4474   \dim_compare:nNnT { \l_@@_y_final_dim } = { \c_max_dim }
4475   {
4476     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4477     \dim_set:Nn \l_@@_y_final_dim
4478     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4479   }
4480 }

```

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.

```

4481 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4482 {
4483   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4484   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4485   {
4486     \@@_find_extremities_of_line:nnnn { #1 } { #2 } { 1 } { 0 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4487   \group_begin:
4488     \@@_open_shorten:
4489     \int_if_zero:nTF { #2 }
4490     { \color { nicematrix-first-col } }
4491     {
4492       \int_compare:nNnT { #2 } = { \l_@@_last_col_int }
4493       { \color { nicematrix-last-col } }
4494     }
4495     \keys_set:nn { nicematrix / xdots } { #3 }
4496     \@@_color:o \l_@@_xdots_color_tl
4497     \bool_if:NTF \l_@@_Vbrace_bool
4498     { \@@_actually_draw_Vbrace: }
4499     { \@@_actually_draw_Vdots: }
4500   \group_end:
4501 }
4502 }

```

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`.


```

4503 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4504 {
4505   \bool_lazy_and:nnTF { \l_@@_initial_open_bool } { \l_@@_final_open_bool }
4506   { \@@_actually_draw_Vdots_i: }
4507   { \@@_actually_draw_Vdots_ii: }
4508   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4509   \@@_draw_line:
4510 }

```

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

```

4511 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4512 {
4513   \@@_open_y_initial_dim:
4514   \@@_open_y_final_dim:
4515   \int_if_zero:nTF { \l_@@_initial_j_int }

```

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

```

4516   {
4517     \@@_qpoint:n { col - 1 }
4518     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4519     \dim_sub:Nn \l_@@_x_initial_dim
4520     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4521   }
4522   {
4523     \bool_lazy_and:nnTF
4524     { \int_compare_p:nNn { \l_@@_last_col_int } > { -2 } }
4525     {
4526       \int_compare_p:nNn
4527       { \l_@@_initial_j_int } = { \g_@@_col_total_int }
4528     }

```

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

```

4529   {
4530     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4531     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4532     \dim_add:Nn \l_@@_x_initial_dim
4533     { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4534   }

```

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

```

4535   {
4536     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4537     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4538     \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4539     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4540   }
4541 }
4542 }

```

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

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

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

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

```

4543 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4544 {
4545   \bool_set_false:N \l_tmpa_bool
4546   \bool_if:NF \l_@@_initial_open_bool
4547   {
4548     \bool_if:NF \l_@@_final_open_bool
4549     {
4550       \@@_set_initial_coords_from_anchor:n { south-west }
4551       \@@_set_final_coords_from_anchor:n { north-west }
4552       \bool_set:Nn \l_tmpa_bool

```

```

4553         {
4554             \dim_compare_p:nNn
4555             { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4556         }
4557     }
4558 }

```

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

```

4559 \bool_if:NTF \l_@@_initial_open_bool
4560 {
4561     \@@_open_y_initial_dim:
4562     \@@_set_final_coords_from_anchor:n { north }
4563     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4564 }
4565 {
4566     \@@_set_initial_coords_from_anchor:n { south }
4567     \bool_if:NTF \l_@@_final_open_bool
4568     { \@@_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.

```

4569     {
4570         \@@_set_final_coords_from_anchor:n { north }
4571         \dim_compare:nNnF { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4572         {
4573             \dim_set:Nn \l_@@_x_initial_dim
4574             {
4575                 \bool_if:NTF \l_tmpa_bool { \dim_min:nn } { \dim_max:nn }
4576                 \l_@@_x_initial_dim \l_@@_x_final_dim
4577             }
4578         }
4579     }
4580 }
4581 }

```

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`.

```

4582 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4583 {
4584     \bool_if:NTF \l_@@_initial_open_bool
4585     { \@@_open_y_initial_dim: }
4586     { \@@_set_initial_coords_from_anchor:n { south } }
4587     \bool_if:NTF \l_@@_final_open_bool
4588     { \@@_open_y_final_dim: }
4589     { \@@_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.

```

4590 \int_if_zero:nTF { \l_@@_initial_j_int }
4591 {
4592     \@@_qpoint:n { col - 1 }

```

```

4593     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4594     \dim_sub:Nn \l_@@_x_initial_dim
4595     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4596 }

```

Elsewhere, the brace must be drawn left flush.

```

4597 {
4598   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4599   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4600   \dim_add:Nn \l_@@_x_initial_dim
4601   { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4602 }

```

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

```

4603   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4604   \@@_draw_line:
4605 }

```

```

4606 \cs_new:Npn \@@_colsep:
4607 { \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.

```

4608 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4609 {
4610   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4611   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4612   {
4613     \@@_find_extremities_of_line:nnnn { #1 } { #2 } { 1 } { 1 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4614     \group_begin:
4615     \@@_open_shorten:
4616     \keys_set:nn { nicematrix / xdots } { #3 }
4617     \@@_color:o \l_@@_xdots_color_tl
4618     \@@_actually_draw_Ddots:
4619   \group_end:
4620 }
4621 }

```

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`.

```

4622 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4623 {
4624   \bool_if:NTF \l_@@_initial_open_bool
4625   {
4626     \@@_open_y_initial_dim:
4627     \@@_open_x_initial_dim:
4628   }
4629   { \@@_set_initial_coords_from_anchor:n { south-east } }
4630   \bool_if:NTF \l_@@_final_open_bool
4631   {
4632     \@@_open_x_final_dim:
4633     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4634   }
4635   { \@@_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.

```

4636   \bool_if:NT \l_@@_parallelize_diags_bool
4637   {
4638     \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).

```

4639     \int_compare:nNnTF { \g_@@_ddots_int } = { \c_one_int }

```

If the diagonal line is the first one, we have no adjustment of the line to do but we store the Δ_x and the Δ_y of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```

4640     {
4641       \dim_gset:Nn \g_@@_delta_x_one_dim
4642       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4643       \dim_gset:Nn \g_@@_delta_y_one_dim
4644       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4645     }

```

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`.

```

4646     {
4647       \dim_compare:nNnF { \g_@@_delta_x_one_dim } = { \c_zero_dim }
4648       {
4649         \dim_set:Nn \l_@@_y_final_dim
4650         {
4651           \l_@@_y_initial_dim +
4652           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4653           \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4654         }
4655       }
4656     }
4657   }
4658   \@@_draw_line:
4659 }

```

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.

```

4660 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4661 {
4662   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4663   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4664   {
4665     \@@_find_extremities_of_line:nnnn { #1 } { #2 } { 1 } { -1 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4666   \group_begin:

```

```

4667         \@@_open_shorten:
4668         \keys_set:nn { nicematrix / xdots } { #3 }
4669         \@@_color:o \l_@@_xdots_color_tl
4670         \@@_actually_draw_Iddots:
4671     \group_end:
4672 }
4673 }

```

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.`

```

4674 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4675 {
4676     \bool_if:NTF \l_@@_initial_open_bool
4677     {
4678         \@@_open_y_initial_dim:
4679         \@@_open_x_initial_dim:
4680     }
4681     { \@@_set_initial_coords_from_anchor:n { south-west } }
4682     \bool_if:NTF \l_@@_final_open_bool
4683     {
4684         \@@_open_y_final_dim:
4685         \@@_open_x_final_dim:
4686     }
4687     { \@@_set_final_coords_from_anchor:n { north-east } }
4688     \bool_if:NT \l_@@_parallelize_diags_bool
4689     {
4690         \int_gincr:N \g_@@_iddots_int
4691         \int_compare:nNnTF { \g_@@_iddots_int } = { \c_one_int }
4692         {
4693             \dim_gset:Nn \g_@@_delta_x_two_dim
4694             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4695             \dim_gset:Nn \g_@@_delta_y_two_dim
4696             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4697         }
4698         {
4699             \dim_compare:nNnF { \g_@@_delta_x_two_dim } = { \c_zero_dim }
4700             {
4701                 \dim_set:Nn \l_@@_y_final_dim
4702                 {
4703                     \l_@@_y_initial_dim +
4704                     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4705                     \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4706                 }
4707             }
4708         }
4709     }
4710     \@@_draw_line:
4711 }

```

17 The actual instructions for drawing the dotted lines with Tikz

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

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

```

4712 \cs_new_protected:Npn \@@_draw_line:
4713 {
4714   \pgfrememberpicturepositiononpagetrue
4715   \pgf@relevantforpicturesizefalse
4716   \bool_lazy_or:nnTF
4717     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4718     { \l_@@_dotted_bool }
4719     { \@@_draw_standard_dotted_line: }
4720     { \@@_draw_unstandard_dotted_line: }
4721 }

```

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.

```

4722 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4723 {
4724   \begin { scope }
4725     \@@_draw_unstandard_dotted_line:o
4726     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4727 }

```

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.

```

4728 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4729 {
4730   \@@_draw_unstandard_dotted_line:nooo
4731   { #1 }
4732   \l_@@_xdots_up_tl
4733   \l_@@_xdots_down_tl
4734   \l_@@_xdots_middle_tl
4735 }
4736 \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.

```

4737 \hook_gput_code:nnn { begindocument } { . }
4738 {
4739   \IfPackageLoadedT { tikz }
4740   {
4741     \tikzset
4742     {
4743       @@_node_above / .style = { sloped , above } ,
4744       @@_node_below / .style = { sloped , below } ,
4745       @@_node_middle / .style =
4746       {

```

```

4747         sloped ,
4748         inner~sep = \c_@@_innersep_middle_dim
4749     }
4750 }
4751 }
4752 }

```

```

4753 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4754 {

```

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.

```

4755     \dim_zero_new:N \l_@@_l_dim
4756     \dim_set:Nn \l_@@_l_dim
4757     {
4758         \fp_to_dim:n
4759         {
4760             sqrt
4761             (
4762                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4763                 +
4764                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4765             )
4766         }
4767     }

```

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.

```

4768     \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4769     {
4770         \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4771         \@@_draw_unstandard_dotted_line_i:
4772     }

```

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

```

4773     \bool_if:NT \l_@@_xdots_h_labels_bool
4774     {
4775         \tikzset
4776         {
4777             @@_node_above / .style = { auto = left } ,
4778             @@_node_below / .style = { auto = right } ,
4779             @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4780         }
4781     }
4782     \tl_if_empty:nF { #4 }
4783     { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4784     \draw
4785     [ #1 ]
4786     ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )

```

Be careful: We can’t put `\c_math_toggle_token` instead of `$` in the following lines because we are in the contents of Tikz nodes (and they will be *rescanned* if the Tikz library `babel` is loaded).

```

4787         -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4788         node [ @@_node_below ] { $ \scriptstyle #3 $ }
4789         node [ @@_node_above ] { $ \scriptstyle #2 $ }
4790         ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4791     \end { scope }
4792 }
4793 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }

```

```

4794 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4795 {
4796   \dim_set:Nn \l_tmpa_dim
4797   {
4798     \l_@@_x_initial_dim
4799     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4800     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4801   }
4802   \dim_set:Nn \l_tmpb_dim
4803   {
4804     \l_@@_y_initial_dim
4805     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4806     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4807   }
4808   \dim_set:Nn \l_@@_tmpc_dim
4809   {
4810     \l_@@_x_final_dim
4811     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4812     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4813   }
4814   \dim_set:Nn \l_@@_tmpd_dim
4815   {
4816     \l_@@_y_final_dim
4817     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4818     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4819   }
4820   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4821   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4822   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4823   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4824 }

```

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

```

4825 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4826 {
4827   \group_begin:

```

The dimension `\l_@@_l_dim` is the length ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

4828   \dim_zero_new:N \l_@@_l_dim
4829   \dim_set:Nn \l_@@_l_dim
4830   {
4831     \fp_to_dim:n
4832     {
4833       sqrt
4834       (
4835         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4836         +
4837         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4838       )
4839     }
4840   }

```

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.

```

4841   \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4842   {
4843     \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4844     { \@@_draw_standard_dotted_line_i: }
4845   }
4846   \group_end:

```



```

4847 \bool_lazy_all:nF
4848 {
4849   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4850   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4851   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4852 }
4853 { \@@_labels_standard_dotted_line: }
4854 }
4855 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4856 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4857 {

```

The number of dots will be $\l_tmpa_int + 1$.

```

4858 \int_set:Nn \l_tmpa_int
4859 {
4860   \dim_ratio:nn
4861   {
4862     \l_@@_l_dim
4863     - \l_@@_xdots_shorten_start_dim
4864     - \l_@@_xdots_shorten_end_dim
4865   }
4866   { \l_@@_xdots_inter_dim }
4867 }

```

The dimensions \l_tmpa_dim and \l_tmpb_dim are the coordinates of the vector between two dots in the dotted line.

```

4868 \dim_set:Nn \l_tmpa_dim
4869 {
4870   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4871   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4872 }
4873 \dim_set:Nn \l_tmpb_dim
4874 {
4875   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4876   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4877 }

```

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

```

4878 \dim_gadd:Nn \l_@@_x_initial_dim
4879 {
4880   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4881   \dim_ratio:nn
4882   {
4883     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4884     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4885   }
4886   { 2 \l_@@_l_dim }
4887 }
4888 \dim_gadd:Nn \l_@@_y_initial_dim
4889 {
4890   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4891   \dim_ratio:nn
4892   {
4893     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4894     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4895   }
4896   { 2 \l_@@_l_dim }
4897 }
4898 \pgf@relevantforpicturesizefalse
4899 \int_step_inline:nnn { \c_zero_int } { \l_tmpa_int }
4900 {
4901   \pgfpathcircle

```

```

4902         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4903         { \l_@@_xdots_radius_dim }
4904         \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4905         \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4906     }
4907     \pgfusepathqfill
4908 }

4909 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
4910 {
4911     \pgfscope
4912     \pgftransformshift
4913     {
4914         \pgfpointlineattime { 0.5 }
4915         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4916         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4917     }
4918     \fp_set:Nn \l_tmpa_fp
4919     {
4920         atand
4921         (
4922             \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4923             \l_@@_x_final_dim - \l_@@_x_initial_dim
4924         )
4925     }
4926     \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4927     \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4928     \tl_if_empty:NF \l_@@_xdots_middle_tl
4929     {
4930         \begin { pgfscope }
4931         \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4932         \pgfnode
4933         { rectangle }
4934         { center }
4935         {
4936             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4937             {
4938                 \c_math_toggle_token
4939                 \scriptstyle \l_@@_xdots_middle_tl
4940                 \c_math_toggle_token
4941             }
4942         }
4943         { }
4944         {
4945             \pgfsetfillcolor { white }
4946             \pgfusepath { fill }
4947         }
4948         \end { pgfscope }
4949     }
4950     \tl_if_empty:NF \l_@@_xdots_up_tl
4951     {
4952         \pgfnode
4953         { rectangle }
4954         { south }
4955         {
4956             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4957             {
4958                 \c_math_toggle_token
4959                 \scriptstyle \l_@@_xdots_up_tl
4960                 \c_math_toggle_token
4961             }
4962         }
4963         { }

```

```

4964         { \pgfusepath { } }
4965     }
4966     \tl_if_empty:NF \l_@@_xdots_down_tl
4967     {
4968         \pgfnode
4969         { rectangle }
4970         { north }
4971         {
4972             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4973             {
4974                 \c_math_toggle_token
4975                 \scriptstyle \l_@@_xdots_down_tl
4976                 \c_math_toggle_token
4977             }
4978         }
4979         { }
4980         { \pgfusepath { } }
4981     }
4982     \endpgfscope
4983 }

```

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 `\hook_gput_code:nnn { begindocument } { . }` and the *arg spec* will be rescanned.

```

4984 \hook_gput_code:nnn { begindocument } { . }
4985 {

```

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

```

4986     \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
4987     \cs_new_protected:Npn \@@_Ldots:
4988     { \@@_collect_options:n { \@@_Ldots_i } }
4989     \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4990     {
4991         \int_if_zero:nTF { \c_jCol }
4992         { \@@_error:nn { in~first~col } { \Ldots } }
4993         {
4994             \int_compare:nNnTF { \c_jCol } = { \l_@@_last_col_int }
4995             { \@@_error:nn { in~last~col } { \Ldots } }
4996             {
4997                 \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
4998                 { #1 , down = #2 , up = #3 , middle = #4 }
4999             }
5000         }
5001         \bool_if:NF \l_@@_nullify_dots_bool
5002         { \phantom { \ensuremath { \@@_old_ldots: } } }
5003         \bool_gset_true:N \g_@@_empty_cell_bool
5004     }

5005     \cs_new_protected:Npn \@@_Cdots:

```

```

5006 { \@@_collect_options:n { \@@_Cdots_i } }
5007 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5008 {
5009   \int_if_zero:nTF { \c@jCol }
5010   { \@@_error:nn { in~first~col } { \Cdots } }
5011   {
5012     \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5013     { \@@_error:nn { in~last~col } { \Cdots } }
5014     {
5015       \@@_instruction_of_type:nnn { \c_false_bool } { Cdots }
5016       { #1 , down = #2 , up = #3 , middle = #4 }
5017     }
5018   }
5019   \bool_if:NF \l_@@_nullify_dots_bool
5020   { \phantom { \ensuremath { \@@_old_cdots: } } }
5021   \bool_gset_true:N \g_@@_empty_cell_bool
5022 }

5023 \cs_new_protected:Npn \@@_Vdots:
5024 { \@@_collect_options:n { \@@_Vdots_i } }
5025 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5026 {
5027   \int_if_zero:nTF { \c@iRow }
5028   { \@@_error:nn { in~first~row } { \Vdots } }
5029   {
5030     \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
5031     { \@@_error:nn { in~last~row } { \Vdots } }
5032     {
5033       \@@_instruction_of_type:nnn { \c_false_bool } { Vdots }
5034       { #1 , down = #2 , up = #3 , middle = #4 }
5035     }
5036   }
5037   \bool_if:NF \l_@@_nullify_dots_bool
5038   { \phantom { \ensuremath { \@@_old_vdots: } } }
5039   \bool_gset_true:N \g_@@_empty_cell_bool
5040 }

5041 \cs_new_protected:Npn \@@_Ddots:
5042 { \@@_collect_options:n { \@@_Ddots_i } }
5043 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5044 {
5045   \int_case:nnF \c@iRow
5046   {
5047     0 { \@@_error:nn { in~first~row } { \Ddots } }
5048     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5049   }
5050   {
5051     \int_case:nnF \c@jCol
5052     {
5053       0 { \@@_error:nn { in~first~col } { \Ddots } }
5054       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5055     }
5056     {
5057       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5058       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5059       { #1 , down = #2 , up = #3 , middle = #4 }
5060     }
5061   }
5062 }
5063 \bool_if:NF \l_@@_nullify_dots_bool
5064 { \phantom { \ensuremath { \@@_old_ddots: } } }
5065 \bool_gset_true:N \g_@@_empty_cell_bool
5066 }

```

```

5067 \cs_new_protected:Npn \@@_Iddots:
5068 { \@@_collect_options:n { \@@_Iddots_i } }
5069 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5070 {
5071   \int_case:nnF \c@iRow
5072   {
5073     0 { \@@_error:nn { in~first~row } { \Iddots } }
5074     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5075   }
5076   {
5077     \int_case:nnF \c@jCol
5078     {
5079       0 { \@@_error:nn { in~first~col } { \Iddots } }
5080       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5081     }
5082     {
5083       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5084       \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { \Iddots }
5085       { #1 , down = #2 , up = #3 , middle = #4 }
5086     }
5087   }
5088   \bool_if:NF \l_@@_nullify_dots_bool
5089   { \phantom { \ensuremath { \@@_old_iddots: } } }
5090   \bool_gset_true:N \g_@@_empty_cell_bool
5091 }
5092 }

```

End of the \AddToHook.

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

```

5093 \keys_define:nn { nicematrix / Ddots }
5094 {
5095   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5096   draw-first .default:n = true ,
5097   draw-first .value_forbidden:n = true
5098 }

```

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

```

5099 \cs_new_protected:Npn \@@_Hspace:
5100 {
5101   \bool_gset_true:N \g_@@_empty_cell_bool
5102   \hspace
5103 }

```

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.

```

5104 \cs_set_eq:NN \@@_old_multicolumn: \multicolumn

```

The command \@@_Hdotsfor will be linked to \Hdotsfor in {NiceArrayWithDelims}. Tikz nodes are created also in the implicit cells of the \Hdotsfor (maybe we should modify that point).

This command must *not* be protected since it begins with \multicolumn.

```

5105 \cs_new:Npn \@@_Hdotsfor:
5106 {
5107   \bool_lazy_and:nnTF
5108   { \int_if_zero_p:n { \c@jCol } }
5109   { \int_if_zero_p:n { \l_@@_first_col_int } }
5110   {
5111     \bool_if:NTF \g_@@_after_col_zero_bool
5112     {
5113       \multicolumn { 1 } { c } { }
5114       \@@_Hdotsfor_i:

```

```

5115     }
5116     { \@@_fatal:n { Hdotsfor~in~col-0 } }
5117 }
5118 {
5119     \multicolumn { 1 } { c } { }
5120     \@@_Hdotsfor_i:
5121 }
5122 }

```

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:`).

```

5123 \hook_gput_code:nnn { begindocument } { . }
5124 {

```

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

```

5125     \cs_new_protected:Npn \@@_Hdotsfor_i:
5126     { \@@_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`).

```

5127     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5128     \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5129     {
5130         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5131         {
5132             \@@_Hdotsfor:nnnn
5133             { \int_use:N \c@iRow }
5134             { \int_use:N \c@jCol }
5135             { #2 }
5136             {
5137                 #1 , #3 ,
5138                 down = \exp_not:n { #4 } ,
5139                 up = \exp_not:n { #5 } ,
5140                 middle = \exp_not:n { #6 }
5141             }
5142         }
5143         \prg_replicate:nn { #2 - 1 }
5144         {
5145             &
5146             \multicolumn { 1 } { c } { }
5147             \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5148         }
5149     }
5150 }

```

```

5151 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5152 {
5153     \bool_set_false:N \l_@@_initial_open_bool
5154     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5155     \int_set:Nn \l_@@_initial_i_int { #1 }
5156     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5157     \int_compare:nNnTF { #2 } = { \c_one_int }
5158     {
5159         \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5160         \bool_set_true:N \l_@@_initial_open_bool
5161     }
5162     {
5163         \cs_if_exist:cTF
5164         {

```

```

5165         pgf @ sh @ ns @ \@@_env:
5166         - \int_use:N \l_@@_initial_i_int
5167         - \int_eval:n { #2 - 1 }
5168     }
5169     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5170     {
5171         \int_set:Nn \l_@@_initial_j_int { #2 }
5172         \bool_set_true:N \l_@@_initial_open_bool
5173     }
5174 }
5175 \int_compare:nNnTF { #2 + #3 - 1 } = { \c@jCol }
5176 {
5177     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5178     \bool_set_true:N \l_@@_final_open_bool
5179 }
5180 {
5181     \cs_if_exist:cTF
5182     {
5183         pgf @ sh @ ns @ \@@_env:
5184         - \int_use:N \l_@@_final_i_int
5185         - \int_eval:n { #2 + #3 }
5186     }
5187     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5188     {
5189         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5190         \bool_set_true:N \l_@@_final_open_bool
5191     }
5192 }
5193 \group_begin:
5194 \@@_open_shorten:
5195 \int_if_zero:nTF { #1 }
5196 { \color { nicematrix-first-row } }
5197 {
5198     \int_compare:nNnT { #1 } = { \g_@@_row_total_int }
5199     { \color { nicematrix-last-row } }
5200 }
5201 \keys_set:nn { nicematrix / xdots } { #4 }
5202 \@@_color:o \l_@@_xdots_color_tl
5203 \@@_actually_draw_Ldots:
5204 \group_end:

```

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

```

5205     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5206     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5207 }

5208 \hook_gput_code:nnn { begindocument } { . }
5209 {
5210     \cs_new_protected:Npn \@@_Vdotsfor:
5211     { \@@_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`).

```

5212     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5213     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5214     {
5215         \bool_gset_true:N \g_@@_empty_cell_bool
5216         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5217         {
5218             \@@_Vdotsfor:nnnn

```

```

5219         { \int_use:N \c@iRow }
5220         { \int_use:N \c@jCol }
5221         { #2 }
5222         {
5223             #1 , #3 ,
5224             down = \exp_not:n { #4 } ,
5225             up = \exp_not:n { #5 } ,
5226             middle = \exp_not:n { #6 }
5227         }
5228     }
5229 }
5230 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5231 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5232 {
5233     \bool_set_false:N \l_@@_initial_open_bool
5234     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5235     \int_set:Nn \l_@@_initial_j_int { #2 }
5236     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5237     \int_compare:nNnTF { #1 } = { \c_one_int }
5238     {
5239         \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5240         \bool_set_true:N \l_@@_initial_open_bool
5241     }
5242     {
5243         \cs_if_exist:cTF
5244         {
5245             pgf @ sh @ ns @ \@@_env:
5246             - \int_eval:n { #1 - 1 }
5247             - \int_use:N \l_@@_initial_j_int
5248         }
5249         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5250         {
5251             \int_set:Nn \l_@@_initial_i_int { #1 }
5252             \bool_set_true:N \l_@@_initial_open_bool
5253         }
5254     }
5255     \int_compare:nNnTF { #1 + #3 - 1 } = { \c@iRow }
5256     {
5257         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5258         \bool_set_true:N \l_@@_final_open_bool
5259     }
5260     {
5261         \cs_if_exist:cTF
5262         {
5263             pgf @ sh @ ns @ \@@_env:
5264             - \int_eval:n { #1 + #3 }
5265             - \int_use:N \l_@@_final_j_int
5266         }
5267         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5268         {
5269             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5270             \bool_set_true:N \l_@@_final_open_bool
5271         }
5272     }

```



```

5273 \group_begin:
5274 \@@_open_shorten:
5275 \int_if_zero:nTF { #2 }
5276 { \color { nicematrix-first-col } }
5277 {
5278   \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
5279   { \color { nicematrix-last-col } }
5280 }
5281 \keys_set:nn { nicematrix / xdots } { #4 }
5282 \@@_color:o \l_@@_xdots_color_tl
5283 \bool_if:NTF \l_@@_Vbrace_bool
5284 { \@@_actually_draw_Vbrace: }
5285 { \@@_actually_draw_Vdots: }
5286 \group_end:

```

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_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5287 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5288 { \cs_set_nopar:cpn { @@_dotted_ ##1 - #2 } { } }
5289 }

```

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

```

5290 \NewDocumentCommand \@@_rotate: { 0 { } }
5291 {
5292   \bool_gset_true:N \g_@@_rotate_bool
5293   \keys_set:nn { nicematrix / rotate } { #1 }
5294   \ignorespaces
5295 }

5296 \keys_define:nn { nicematrix / rotate }
5297 {
5298   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5299   c .value_forbidden:n = true ,
5300   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5301 }

```

19 The command `\line` accessible in code-after

In the `\CodeAfter`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format i - j) and draws a dotted line between these cells. In fact, it also works with names of blocks.

First, we write a command with the following behaviour:

- If the argument is of the format i - j , our command applies the command `\int_eval:n` to i and j ;
- If not (that is to say, when it’s a name of a `\Block`), the argument is left unchanged.

This must *not* be protected (and is, of course fully expandable).¹⁴

```

5302 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop

```

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

```

5303 {
5304   \tl_if_empty:nTF { #2 }
5305     { #1 }
5306     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5307 }
5308 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5309 { \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).

```

5310 \hook_gput_code:nnn { begindocument } { . }
5311 {

```

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

```

5312   \tl_set_rescan:Nnn \l_tmpa_tl { }
5313   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5314   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5315     {
5316       \group_begin:
5317       \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5318       \@@_color:o \l_@@_xdots_color_tl
5319       \use:e
5320       {
5321         \@@_line_i:nn
5322         { \@@_double_int_eval:n #2 - \q_stop }
5323         { \@@_double_int_eval:n #3 - \q_stop }
5324       }
5325       \group_end:
5326     }
5327 }
5328 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5329 {
5330   \bool_set_false:N \l_@@_initial_open_bool
5331   \bool_set_false:N \l_@@_final_open_bool
5332   \bool_lazy_or:nnTF
5333     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5334     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5335     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

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

```

5336   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5337 }
5338 \hook_gput_code:nnn { begindocument } { . }
5339 {
5340   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5341   {

```

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:`.

```

5342     \c_@@_pgfortikzpicture_tl
5343     \@@_draw_line_iii:nn { #1 } { #2 }
5344     \c_@@_endpgfortikzpicture_tl
5345   }
5346 }

```

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

```

5347 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5348 {
5349   \pgfrememberpicturepositiononpagetrue

```

```

5350 \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5351 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5352 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5353 \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5354 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5355 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5356 \@@_draw_line:
5357 }

```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don't use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

20 The command `\RowStyle`

`\g_@@_row_style_tl` may contain several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

Then, `\g_@@_row_style_tl` will be inserted in all the cells of the array (and also in both components of a `\diagbox` in a cell of in a mono-row block).

The test `\@@_if_row_less_than:nn` ensures that the instructions are inserted only if you are in a row which is (still) in the scope of that instructions (which depends on the value of the key `nb-rows` of `\RowStyle`).

That test will be active even in an expandable context because `\@@_if_row_less_than:nn` is *not* protected.

`#1` is the first row *after* the scope of the instructions in `#2`

However, both arguments are implicit because they are taken by curryfication.

```

5358 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNnT { \c@iRow } < }
5359 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNnF { \c@jCol } < }

```

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

```

5360 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5361 {
5362   \tl_gput_right:Ne \g_@@_row_style_tl
5363   {

```

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

```

5364 \exp_not:N
5365 \@@_if_row_less_than:nn
5366 { \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`).

```

5367 {
5368   \exp_not:N
5369   \@@_if_col_greater_than:nn
5370   { \int_eval:n { \c@jCol } }
5371   { \exp_not:n { #1 } \scan_stop: }
5372 }
5373 }
5374 }
5375 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

```

```

5376 \keys_define:nn { nicematrix / RowStyle }
5377 {
5378   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5379   cell-space-top-limit .value_required:n = true ,
5380   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5381   cell-space-bottom-limit .value_required:n = true ,

```

```

5382 cell-space-limits .meta:n =
5383 {
5384     cell-space-top-limit = #1 ,
5385     cell-space-bottom-limit = #1 ,
5386 } ,
5387 color .tl_set:N = \l_@@_color_tl ,
5388 color .value_required:n = true ,
5389 bold .bool_set:N = \l_@@_bold_row_style_bool ,
5390 bold .default:n = true ,
5391 nb-rows .code:n =
5392     \str_if_eq:eeTF { #1 } { * }
5393     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5394     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5395 nb-rows .value_required:n = true ,
5396 fill .tl_set:N = \l_@@_fill_tl ,
5397 fill .value_required:n = true ,
5398 opacity .tl_set:N = \l_@@_opacity_tl ,
5399 opacity .value_required:n = true ,
5400 rowcolor .tl_set:N = \l_@@_fill_tl ,
5401 rowcolor .value_required:n = true ,
5402 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5403 rounded-corners .default:n = 4 pt ,
5404 unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5405 }

```

```

5406 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5407 {
5408     \group_begin:
5409     \tl_clear:N \l_@@_fill_tl
5410     \tl_clear:N \l_@@_opacity_tl
5411     \tl_clear:N \l_@@_color_tl
5412     \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5413     \dim_zero:N \l_@@_rounded_corners_dim
5414     \dim_zero:N \l_tmpa_dim
5415     \dim_zero:N \l_tmpb_dim
5416     \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5417     \tl_if_empty:NF \l_@@_fill_tl
5418     {
5419         \@@_add_opacity_to_fill:
5420         \tl_gput_right:Ne \g_@@_pre_code_before_tl
5421         {

```

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

```

5422         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5423         { \int_use:N \c_iRow - \int_use:N \c_jCol }
5424         {
5425             \int_eval:n { \c_iRow + \l_@@_key_nb_rows_int - 1 }
5426             - *
5427         }
5428         { \dim_use:N \l_@@_rounded_corners_dim }
5429     }
5430 }
5431 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5432     \dim_compare:nNnT { \l_tmpa_dim } > { \c_zero_dim }
5433     {
5434         \@@_put_in_row_style:e
5435         {
5436             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5437             {

```

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

```

5438         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5439         { \dim_use:N \l_tmpa_dim }
5440     }
5441 }
5442 }

```

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

```

5443 \dim_compare:nNnT { \l_tmpb_dim } > { \c_zero_dim }
5444 {
5445     \@@_put_in_row_style:e
5446     {
5447         \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5448         {
5449             \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5450             { \dim_use:N \l_tmpb_dim }
5451         }
5452     }
5453 }

```

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

```

5454 \tl_if_empty:NF \l_@@_color_tl
5455 {
5456     \@@_put_in_row_style:e
5457     {
5458         \mode_leave_vertical:
5459         \@@_color:n { \l_@@_color_tl }
5460     }
5461 }

```

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

```

5462 \bool_if:NT \l_@@_bold_row_style_bool
5463 {
5464     \@@_put_in_row_style:n
5465     {
5466         \exp_not:n
5467         {
5468             \if_mode_math:
5469             \c_math_toggle_token
5470             \bfseries \boldmath
5471             \c_math_toggle_token
5472         \else:
5473             \bfseries \boldmath
5474         \fi:
5475     }
5476 }
5477 }
5478 \group_end:
5479 \g_@@_row_style_tl
5480 \ignorespaces
5481 }

```

The following commande must *not* be protected.

```

5482 \cs_new:Npn \@@_rounded_from_row:n #1
5483 {
5484     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

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

```

5485     { \int_eval:n { #1 } - 1 }
5486     {
5487         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5488         - \exp_not:n { \int_use:N \c@jCol }
5489     }
5490     { \dim_use:N \l_@@_rounded_corners_dim }
5491 }

```

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).

```
5492 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5493 {
```

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.

```
5494 \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`.

```
5495 \str_if_in:nnF { #1 } { !! }
5496 {
5497 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

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

```
5498 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5499 }
5500 \int_if_zero:nTF { \l_tmpa_int }
```

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

```
5501 {
5502 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5503 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5504 }
```

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

```
5505 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5506 }
5507 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5508 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

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

```
5509 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5510 {
5511 \dim_compare:nNnT { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim }
5512 {
```

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

```

5513     \group_begin:
5514     \pgfsetcornersarced
5515     {
5516         \pgfpoint
5517         { \l_@@_tab_rounded_corners_dim }
5518         { \l_@@_tab_rounded_corners_dim }
5519     }

```

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.

```

5520     \bool_if:NTF \l_@@_hvlines_bool
5521     {
5522         \pgfpathrectanglecorners
5523         {
5524             \pgfpointadd
5525             { \@@_qpoint:n { row-1 } }
5526             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5527         }
5528         {
5529             \pgfpointadd
5530             {
5531                 \@@_qpoint:n
5532                 { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5533             }
5534             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5535         }
5536     }
5537     {
5538         \pgfpathrectanglecorners
5539         { \@@_qpoint:n { row-1 } }
5540         {
5541             \pgfpointadd
5542             {
5543                 \@@_qpoint:n
5544                 { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5545             }
5546             { \pgfpoint \c_zero_dim \arrayrulewidth }
5547         }
5548     }
5549     \pgfusepath { clip }
5550     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5551     }
5552 }

```

The macro `\@@_actually_color:` will actually fill all the rectangles, color by color (using the sequence `\l_@@_colors_seq` and all the token lists of the form `\l_@@_color_i_tl`).

```

5553 \cs_new_protected:Npn \@@_actually_color:
5554 {
5555     \pgfpicture
5556     \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.

```

5557     \@@_clip_with_rounded_corners:
5558     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5559     {
5560         \int_compare:nNnTF { ##1 } = { \c_one_int }

```

```

5561     {
5562         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5563         \use:c { g_@@_color _ 1 _tl }
5564         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5565     }
5566     {
5567         \begin { pgfscope }
5568             \@@_color_opacity: ##2
5569             \use:c { g_@@_color _ ##1 _tl }
5570             \tl_gclear:c { g_@@_color _ ##1 _tl }
5571             \pgfusepath { fill }
5572         \end { pgfscope }
5573     }
5574 }
5575 \endpgfpicture
5576 }

```

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

```

5577 \cs_new_protected:Npn \@@_color_opacity:
5578 {
5579     \peek_meaning:NTF [
5580         { \@@_color_opacity:w }
5581         { \@@_color_opacity:w [ ] }
5582     }

```

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 currying.

```

5583 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5584 {
5585     \tl_clear:N \l_tmpa_tl
5586     \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.

```

5587     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5588     \tl_if_empty:NTF \l_tmpb_tl
5589         { \@declaredcolor }
5590         { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5591 }

```

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

```

5592 \keys_define:nn { nicematrix / color-opacity }
5593 {
5594     opacity .tl_set:N          = \l_tmpa_tl ,
5595     opacity .value_required:n = true
5596 }

```

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

```

5597 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5598 {
5599     \def \l_@@_rows_tl { #1 }
5600     \def \l_@@_cols_tl { #2 }
5601     \@@_cartesian_path:
5602 }

```

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

```

5603 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5604 {
5605     \tl_if_blank:nF { #2 }

```



```

5606 {
5607   \@@_add_to_colors_seq:en
5608   { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5609   { \@@_cartesian_color:nn { #3 } { - } }
5610 }
5611 }

```

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

```

5612 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5613 {
5614   \tl_if_blank:nF { #2 }
5615   {
5616     \@@_add_to_colors_seq:en
5617     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5618     { \@@_cartesian_color:nn { - } { #3 } }
5619   }
5620 }

```

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

```

5621 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5622 {
5623   \tl_if_blank:nF { #2 }
5624   {
5625     \@@_add_to_colors_seq:en
5626     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5627     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5628   }
5629 }

```

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

```

5630 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5631 {
5632   \tl_if_blank:nF { #2 }
5633   {
5634     \@@_add_to_colors_seq:en
5635     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5636     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5637   }
5638 }

```

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

```

5639 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5640 {
5641   \@@_cut_on_hyphen:w #1 \q_stop
5642   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5643   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5644   \@@_cut_on_hyphen:w #2 \q_stop
5645   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5646   \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`.

```

5647   \@@_cartesian_path:n { #3 }
5648 }

```

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

```

5649 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5650 {
5651   \clist_map_inline:nn { #3 }
5652   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5653 }

```

```

5654 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5655 {
5656   \int_step_inline:nn { \c@iRow }
5657   {
5658     \int_step_inline:nn { \c@jCol }
5659     {
5660       \int_if_even:nTF { #####1 + ##1 }
5661       { \@@_cellcolor [ #1 ] { #2 } }
5662       { \@@_cellcolor [ #1 ] { #3 } }
5663       { ##1 - #####1 }
5664     }
5665   }
5666 }

```

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”.

```

5667 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5668 {
5669   \@@_rectanglecolor [ #1 ] { #2 }
5670   { 1 - 1 }
5671   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5672 }

```

```

5673 \keys_define:nn { nicematrix / rowcolors }
5674 {
5675   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5676   respect-blocks .default:n = true ,
5677   cols .tl_set:N = \l_@@_cols_tl ,
5678   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5679   restart .default:n = true ,
5680   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5681 }

```

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*.

```

5682 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5683 {

```

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

```

5684   \group_begin:
5685   \seq_clear_new:N \l_@@_colors_seq
5686   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5687   \tl_clear_new:N \l_@@_cols_tl
5688   \tl_set:Nn \l_@@_cols_tl { - }
5689   \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).

```

5690   \int_zero_new:N \l_@@_color_int
5691   \int_set_eq:NN \l_@@_color_int \c_one_int
5692   \bool_if:NT \l_@@_respect_blocks_bool
5693   {

```

We don't want to take into account a block which is completely in the “first column” (number 0) or in the “last column” and that's why we filter the sequence of the blocks (in the sequence `\l_tmpa_seq`).

```

5694     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5695     \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5696     { \@@_not_in_exterior_p:nnnnn ##1 }
5697   }
5698   \pgfpicture
5699   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5700   \clist_map_inline:nn { #2 }
5701   {
5702     \tl_set:Nn \l_tmpa_tl { ##1 }
5703     \tl_if_in:NnTF \l_tmpa_tl { - }
5704     { \@@_cut_on_hyphen:w ##1 \q_stop }
5705     { \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.

```

5706     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5707     \int_set:Nn \l_@@_color_int
5708     { \bool_if:NNTF \l_@@_rowcolors_restart_bool { 1 } { \l_tmpa_tl } }
5709     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5710     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5711     {

```

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

```

5712         \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).

```

5713         \bool_if:NT \l_@@_respect_blocks_bool
5714         {
5715           \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5716           { \@@_intersect_our_row_p:nnnnn #####1 }
5717           \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

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

```

5718         }
5719         \tl_set:Nn \l_@@_rows_tl
5720         { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

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

```

5721         \tl_set:Nn \l_@@_color_tl
5722         {
5723           \@@_color_index:n
5724           {
5725             \int_mod:nn
5726             { \l_@@_color_int - 1 }
5727             { \seq_count:N \l_@@_colors_seq }
5728             + 1
5729           }
5730         }
5731         \tl_if_empty:NF \l_@@_color_tl
5732         {
5733           \@@_add_to_colors_seq:ee
5734           { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5735           { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5736         }
5737         \int_incr:N \l_@@_color_int
5738         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5739       }
5740     }
5741   \endpgfpicture
5742   \group_end:
5743 }

```

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.

```
5744 \cs_new:Npn \@@_color_index:n #1
5745 {
```

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

```
5746 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5747 { \@@_color_index:n { #1 - 1 } }
5748 { \seq_item:Nn \l_@@_colors_seq { #1 } }
5749 }
```

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

```
5750 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5751 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }
```

The braces around `#3` and `#4` are mandatory.

```
5752 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5753 {
5754   \int_compare:nNtT { #3 } > { \l_tmpb_int }
5755   { \int_set:Nn \l_tmpb_int { #3 } }
5756 }
```

```
5757 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5758 {
5759   \int_if_zero:nTF { #4 }
5760   { \prg_return_false: }
5761   {
5762     \int_compare:nNtTF { #2 } > { \c@jCol }
5763     { \prg_return_false: }
5764     { \prg_return_true: }
5765   }
5766 }
```

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

```
5767 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5768 {
5769   \int_compare:nNtTF { #1 } > { \l_tmpa_int }
5770   { \prg_return_false: }
5771   {
5772     \int_compare:nNtTF { \l_tmpa_int } > { #3 }
5773     { \prg_return_false: }
5774     { \prg_return_true: }
5775   }
5776 }
```

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`).

```
5777 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5778 {
5779   \dim_compare:nNtTF { #1 } = { \c_zero_dim }
5780   {
5781     \bool_if:NtF \l_@@_nocolor_used_bool
5782     { \@@_cartesian_path_normal_ii: }
```

```

5783     {
5784         \clist_if_empty:NTF \l_@@_corners_cells_clist
5785         { \@@_cartesian_path_normal_i:n { #1 } }
5786         { \@@_cartesian_path_normal_ii: }
5787     }
5788 }
5789 { \@@_cartesian_path_normal_i:n { #1 } }
5790 }

```

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.

```

5791 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5792 {
5793     \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5794     \clist_map_inline:Nn \l_@@_cols_tl
5795     {

```

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

```

5796         \def \l_tmpa_tl { ##1 }
5797         \tl_if_in:NnTF \l_tmpa_tl { - }
5798         { \@@_cut_on_hyphen:w ##1 \q_stop }
5799         { \def \l_tmpb_tl { ##1 } } % 2025-04-16
5800         \tl_if_empty:NTF \l_tmpa_tl
5801         { \def \l_tmpa_tl { 1 } }
5802         {
5803             \str_if_eq:eeT { \l_tmpa_tl } { * }
5804             { \def \l_tmpa_tl { 1 } }
5805         }
5806         \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_col_total_int }
5807         { \@@_error:n { Invalid~col~number } }
5808         \tl_if_empty:NTF \l_tmpb_tl
5809         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5810         {
5811             \str_if_eq:eeT { \l_tmpb_tl } { * }
5812             { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5813         }
5814         \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_col_total_int }
5815         { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

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

```

5816         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5817         \@@_qpoint:n { col - \l_tmpa_tl }
5818         \int_compare:nNnTF { \l_@@_first_col_int } = { \l_tmpa_tl }
5819         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5820         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5821         \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5822         \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.

```

5823     \clist_map_inline:Nn \l_@@_rows_tl
5824     {
5825         \def \l_tmpa_tl { #####1 }
5826         \tl_if_in:NnTF \l_tmpa_tl { - }
5827         { \@@_cut_on_hyphen:w #####1 \q_stop }
5828         { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5829         \tl_if_empty:NTF \l_tmpa_tl
5830         { \def \l_tmpa_tl { 1 } }
5831         {
5832             \str_if_eq:eeT { \l_tmpa_tl } { * }
5833             { \def \l_tmpa_tl { 1 } }
5834         }
5835         \tl_if_empty:NTF \l_tmpb_tl

```

```

5836         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5837     {
5838         \str_if_eq:eeT { \l_tmpb_tl } { * }
5839         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5840     }
5841     \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_row_total_int }
5842     { \@@_error:n { Invalid-row-number } }
5843     \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_row_total_int }
5844     { \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`.

```

5845     \cs_if_exist:cF
5846     { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5847     {
5848         \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5849         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5850         \@@_qpoint:n { row - \l_tmpa_tl }
5851         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5852         \pgfpathrectanglecorners
5853         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5854         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5855     }
5856 }
5857 }
5858 }

```

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

```

5859 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5860 {
5861     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5862     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5863     \clist_map_inline:Nn \l_@@_cols_tl
5864     {
5865         \@@_qpoint:n { col - ##1 }
5866         \int_compare:nNnTF { \l_@@_first_col_int } = { ##1 }
5867         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5868         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5869         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5870         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5871     \clist_map_inline:Nn \l_@@_rows_tl
5872     {
5873         \@@_if_in_corner:nF { #####1 - ##1 }
5874         {
5875             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5876             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5877             \@@_qpoint:n { row - #####1 }
5878             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5879             \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5880             {
5881                 \pgfpathrectanglecorners
5882                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5883                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5884             }
5885         }
5886     }
5887 }
5888 }

```

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`).

```
5889 \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

```
5890 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5891 {
5892   \bool_set_true:N \l_@@_nocolor_used_bool
5893   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5894   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
```

We begin the loop over the columns.

```
5895   \clist_map_inline:Nn \l_@@_rows_tl
5896   {
5897     \clist_map_inline:Nn \l_@@_cols_tl
5898     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - ####1 } { } }
5899   }
5900 }
```

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`.

```
5901 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5902 {
5903   \clist_set_eq:NN \l_tmpa_clist #1
5904   \clist_clear:N #1
5905   \clist_map_inline:Nn \l_tmpa_clist
5906   {
```

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

```
5907     \def \l_tmpa_tl { ##1 }
5908     \tl_if_in:NnTF \l_tmpa_tl { - }
5909     { \@@_cut_on_hyphen:w ##1 \q_stop }
5910     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5911     \bool_lazy_or:nnT
5912     { \str_if_eq_p:ee { \l_tmpa_tl } { * } }
5913     { \tl_if_blank_p:o \l_tmpa_tl }
5914     { \def \l_tmpa_tl { 1 } }
5915     \bool_lazy_or:nnT
5916     { \str_if_eq_p:ee { \l_tmpb_tl } { * } }
5917     { \tl_if_blank_p:o \l_tmpb_tl }
5918     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5919     \int_compare:nNnT { \l_tmpb_tl } > { #2 }
5920     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5921     \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
5922     { \clist_put_right:Nn #1 { ####1 } }
5923   }
5924 }
```

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

```
5925 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5926 {
5927   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5928   {
```

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`).

```
5929     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5930     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5931   }
5932   \ignorespaces
5933 }
```

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

```

5934 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5935 {
5936   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5937   {
5938     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5939     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5940     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5941   }
5942   \ignorespaces
5943 }

```

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

```

5944 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5945 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

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

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

```

5946 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5947 {

```

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`.

```

5948   \seq_gclear:N \g_tmpa_seq
5949   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5950     { \@@_rowlistcolors_tabular:nnnn #1 }
5951   \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`.

```

5952   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5953   {
5954     { \int_use:N \c@iRow }
5955     { \exp_not:n { #1 } }
5956     { \exp_not:n { #2 } }
5957     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5958   }
5959   \ignorespaces
5960 }

```

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`).

```

5961 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
5962 {
5963   \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).

```

5964   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5965   {
5966     \tl_gput_right:Ne \g_@@_pre_code_before_tl

```



```

5967     {
5968         \@_rowlistcolors
5969         [ \exp_not:n { #2 } ]
5970         { #1 - \int_eval:n { \c@iRow - 1 } }
5971         { \exp_not:n { #3 } }
5972         [ \exp_not:n { #4 } ]
5973     }
5974 }
5975 }

```

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.

```

5976 \cs_new_protected:Npn \@_clear_rowlistcolors_seq:
5977 {
5978     \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5979     { \@_rowlistcolors_tabular_ii:nnnn ##1 }
5980     \seq_gclear:N \g_@@_rowlistcolors_seq
5981 }

5982 \cs_new_protected:Npn \@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5983 {
5984     \tl_gput_right:Nn \g_@@_pre_code_before_tl
5985     { \@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5986 }

```

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.

```

5987 \NewDocumentCommand \@_columncolor_preamble { 0 { } m }
5988 {

```

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).

```

5989     \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
5990     {

```

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).

```

5991     \tl_gput_left:Ne \g_@@_pre_code_before_tl
5992     {
5993         \exp_not:N \columncolor [ #1 ]
5994         { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5995     }
5996 }
5997 }

5998 \cs_new_protected:Npn \@_EmptyColumn:n #1
5999 {
6000     \clist_map_inline:nn { #1 }
6001     {
6002         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6003         { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6004         \columncolor { nocolor } { ##1 }
6005     }
6006 }

```

```

6007 \cs_new_protected:Npn \@@_EmptyRow:n #1
6008 {
6009   \clist_map_inline:nn { #1 }
6010   {
6011     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6012     { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6013     \rowcolor { nocolor } { ##1 }
6014   }
6015 }

```

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`.

```

6016 \cs_set_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`.

```

6017 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6018 {
6019   \int_if_zero:nTF { \l_@@_first_col_int }
6020   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6021   {
6022     \int_if_zero:nTF { \c_jCol }
6023     {
6024       \int_compare:nNnF { \c_iRow } = { -1 }
6025       {
6026         \int_compare:nNnF { \c_iRow } = { \l_@@_last_row_int - 1 }
6027         { #1 }
6028       }
6029     }
6030     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6031   }
6032 }

```

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.

```

6033 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6034 {
6035   \int_if_zero:nF { \c_iRow }
6036   {
6037     \int_compare:nNnF { \c_iRow } = { \l_@@_last_row_int }
6038     {
6039       \int_compare:nNnT { \c_jCol } > { \c_zero_int }
6040       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6041     }
6042   }
6043 }

```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to `-2` or `-1` (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

The following command will be used for `\Toprule`, `\BottomRule` and `\MidRule`.

```

6044 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6045 {
6046   \IfPackageLoadedTF { tikz }
6047   {
6048     \IfPackageLoadedTF { booktabs }
6049     { #2 }
6050     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6051   }
6052   { \@@_error:nn { TopRule~without~tikz } { #1 } }
6053 }

6054 \NewExpandableDocumentCommand { \@@_TopRule } { } { }
6055 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }

6056 \cs_new:Npn \@@_TopRule_i:
6057 {
6058   \noalign \bgroup
6059   \peek_meaning:NTF [
6060   { \@@_TopRule_ii: }
6061   { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6062 }

6063 \NewDocumentCommand \@@_TopRule_ii: { o }
6064 {
6065   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6066   {
6067     \@@_hline:n
6068     {
6069       position = \int_eval:n { \c@iRow + 1 } ,
6070       tikz =
6071       {
6072         line-width = #1 ,
6073         yshift = 0.25 \arrayrulewidth ,
6074         shorten-< = - 0.5 \arrayrulewidth
6075       } ,
6076       total-width = #1
6077     }
6078   }
6079   \skip_vertical:n { \belowrulesep + #1 }
6080   \egroup
6081 }

6082 \NewExpandableDocumentCommand { \@@_BottomRule } { } { }
6083 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }

6084 \cs_new:Npn \@@_BottomRule_i:
6085 {
6086   \noalign \bgroup
6087   \peek_meaning:NTF [
6088   { \@@_BottomRule_ii: }
6089   { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6090 }

6091 \NewDocumentCommand \@@_BottomRule_ii: { o }
6092 {
6093   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6094   {
6095     \@@_hline:n
6096     {
6097       position = \int_eval:n { \c@iRow + 1 } ,
6098       tikz =
6099       {
6100         line-width = #1 ,

```

```

6101         yshift = 0.25 \arrayrulewidth ,
6102         shorten~< = - 0.5 \arrayrulewidth
6103     } ,
6104     total-width = #1 ,
6105 }
6106 }
6107 \skip_vertical:N \aboverulesep
6108 \@@_create_row_node_i:
6109 \skip_vertical:n { #1 }
6110 \egroup
6111 }
6112 \NewExpandableDocumentCommand { \@@_MidRule } { }
6113 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6114 \cs_new:Npn \@@_MidRule_i:
6115 {
6116     \noalign \bgroup
6117     \peek_meaning:NTF [
6118         { \@@_MidRule_ii: }
6119         { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6120     ]
6121 \NewDocumentCommand \@@_MidRule_ii: { o }
6122 {
6123     \skip_vertical:N \aboverulesep
6124     \@@_create_row_node_i:
6125     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6126     {
6127         \@@_hline:n
6128         {
6129             position = \int_eval:n { \c@iRow + 1 } ,
6130             tikz =
6131             {
6132                 line-width = #1 ,
6133                 yshift = 0.25 \arrayrulewidth ,
6134                 shorten~< = - 0.5 \arrayrulewidth
6135             } ,
6136             total-width = #1 ,
6137         }
6138     }
6139     \skip_vertical:n { \belowrulesep + #1 }
6140     \egroup
6141 }

```

General system for drawing rules

When a command, environment or “subsystem” of `nicematrix` wants to draw a rule, it will write in the internal `\CodeAfter` a command `\@@_vline:n` or `\@@_hline:n`. Both commands take in as argument a list of *key=value* pairs. That list will first be analyzed with the following set of keys. However, unknown keys will be analyzed further with another set of keys.

```

6142 \keys_define:nn { nicematrix / Rules }
6143 {
6144     position .int_set:N = \l_@@_position_int ,
6145     position .value_required:n = true ,
6146     start .int_set:N = \l_@@_start_int ,
6147     end .code:n =
6148         \bool_lazy_or:nnTF
6149         { \tl_if_empty_p:n { #1 } }
6150         { \str_if_eq_p:ee { #1 } { last } }
6151         { \int_set_eq:NN \l_@@_end_int \c@jCol }
6152         { \int_set:Nn \l_@@_end_int { #1 } }
6153 }

```

It's possible that the rule won't be drawn continuously from `start` to `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That's why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```

6154 \keys_define:nn { nicematrix / RulesBis }
6155   {
6156     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6157     multiplicity .initial:n = 1 ,
6158     dotted .bool_set:N = \l_@@_dotted_bool ,
6159     dotted .initial:n = false ,
6160     dotted .default:n = true ,

```

We want that, even when the rule has been defined with TikZ by the key `tikz`, the user has still the possibility to change the color of the rule with the key `color` (in the command `\Hline`, not in the key `tikz` of the command `\Hline`). The main use is, when the user has defined its own command `\MyDashedLine` by `\newcommand{\MyDashedRule}{\Hline[tikz=dashed]}`, to give the ability to write `\MyDashedRule[color=red]`.

```

6161   color .code:n =
6162     \@@_set_CTarc:n { #1 }
6163     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6164   color .value_required:n = true ,
6165   sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6166   sep-color .value_required:n = true ,

```

If the user uses the key `tikz`, the rule (or more precisely: the different sub-rules since a rule may be broken by blocks or others) will be drawn with Tikz.

```

6167   tikz .code:n =
6168     \IfPackageLoadedTF { tikz }
6169     { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6170     { \@@_error:n { tikz~without~tikz } } ,
6171   tikz .value_required:n = true ,
6172   total-width .dim_set:N = \l_@@_rule_width_dim ,
6173   total-width .value_required:n = true ,
6174   width .meta:n = { total-width = #1 } ,
6175   unknown .code:n = \@@_error:n { Unknown~key~for~RulesBis }
6176 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6177 \cs_new_protected:Npn \@@_vline:n #1
6178   {

```

The group is for the options.

```

6179   \group_begin:
6180   \int_set_eq:NN \l_@@_end_int \c@iRow
6181   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl

```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```

6182   \int_compare:nNnT { \l_@@_position_int } < { \c@jCol + 2 }
6183     \@@_vline_i:
6184   \group_end:
6185 }
6186 \cs_new_protected:Npn \@@_vline_i:
6187   {

```

`\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`.

```

6188   \tl_set:Nn \l_tmpb_tl { \int_use:N \l_@@_position_int }
6189   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int

```

```

6190 \l_tmpa_tl
6191 {

```

The boolean `\g_tmpa_bool` indicates whether the small vertical rule will be drawn. If we find that it is in a block (a real block, created by `\Block` or a virtual block corresponding to a dotted line, created by `\Cdots`, `\Vdots`, etc.), we will set `\g_tmpa_bool` to false and the small vertical rule won't be drawn.

```

6192 \bool_gset_true:N \g_tmpa_bool
6193 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6194 { \@@_test_vline_in_block:nnnnn ##1 }
6195 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6196 { \@@_test_vline_in_block:nnnnn ##1 }
6197 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6198 { \@@_test_vline_in_stroken_block:nnnn ##1 }
6199 \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_v: }
6200 \bool_if:NTF \g_tmpa_bool
6201 {
6202 \int_if_zero:nT { \l_@@_local_start_int }

```

We keep in memory that we have a rule to draw. `\l_@@_local_start_int` will be the starting row of the rule that we will have to draw.

```

6203 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6204 }
6205 {
6206 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6207 {
6208 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6209 \@@_vline_ii:
6210 \int_zero:N \l_@@_local_start_int
6211 }
6212 }
6213 }
6214 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6215 {
6216 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6217 \@@_vline_ii:
6218 }
6219 }

```

```

6220 \cs_new_protected:Npn \@@_test_in_corner_v:
6221 {
6222 \int_compare:nNnTF { \l_tmpb_tl } = { \c_jCol + 1 }
6223 {
6224 \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6225 { \bool_set_false:N \g_tmpa_bool }
6226 }
6227 {
6228 \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6229 {
6230 \int_compare:nNnTF { \l_tmpb_tl } = { \c_one_int }
6231 { \bool_set_false:N \g_tmpa_bool }
6232 {
6233 \@@_if_in_corner:nT
6234 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6235 { \bool_set_false:N \g_tmpa_bool }
6236 }
6237 }
6238 }
6239 }

```

```

6240 \cs_new_protected:Npn \@@_vline_ii:
6241 {

```

```

6242 \tl_clear:N \l_@@_tikz_rule_tl
6243 \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6244 \bool_if:NTF \l_@@_dotted_bool
6245 { \@@_vline_iv: }
6246 {
6247   \tl_if_empty:NTF \l_@@_tikz_rule_tl
6248   { \@@_vline_iii: }
6249   { \@@_vline_v: }
6250 }
6251 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

6252 \cs_new_protected:Npn \@@_vline_iii:
6253 {
6254   \pgfpicture
6255   \pgfrememberpicturepositiononpagetrue
6256   \pgf@relevantforpicturesizefalse
6257   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6258   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6259   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6260   \dim_set:Nn \l_tmpb_dim
6261   {
6262     \pgf@x
6263     - 0.5 \l_@@_rule_width_dim
6264     +
6265     ( \arrayrulewidth * \l_@@_multiplicity_int
6266       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6267   }
6268   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6269   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6270   \bool_lazy_all:nT
6271   {
6272     { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6273     { \cs_if_exist_p:N \CT@drsc@ }
6274     { ! \tl_if_blank_p:o \CT@drsc@ }
6275   }
6276   {
6277     \group_begin:
6278     \CT@drsc@
6279     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6280     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6281     \dim_set:Nn \l_@@_tmpd_dim
6282     {
6283       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6284       * ( \l_@@_multiplicity_int - 1 )
6285     }
6286     \pgfpathrectanglecorners
6287     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6288     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6289     \pgfusepath { fill }
6290     \group_end:
6291   }
6292   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6293   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6294   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6295   {
6296     \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6297     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6298     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6299   }
6300   \CT@arc@
6301   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6302   \pgfsetrectcap

```

```

6303 \pgfusepathqstroke
6304 \endpgfpicture
6305 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6306 \cs_new_protected:Npn \@@_vline_iv:
6307 {
6308   \pgfpicture
6309   \pgfrememberpicturepositiononpagetrue
6310   \pgf@relevantforpicturesizefalse
6311   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6312   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6313   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6314   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6315   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6316   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6317   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6318   \CT@arc@
6319   \@@_draw_line:
6320   \endpgfpicture
6321 }

```

The following code is for the case when the user uses the key `tikz`.

```

6322 \cs_new_protected:Npn \@@_vline_v:
6323 {
6324   \begin { tikzpicture }

```

By default, the color defined by `\arrayrulecolor` or by `rules/color` will be used, but it's still possible to change the color by using the key `color` or, of course, the key `color` inside the key `tikz` (that is to say the key `color` provided by PGF).

```

6325   \CT@arc@
6326   \tl_if_empty:NF \l_@@_rule_color_tl
6327   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6328   \pgfrememberpicturepositiononpagetrue
6329   \pgf@relevantforpicturesizefalse
6330   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6331   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6332   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6333   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6334   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6335   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6336   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6337   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6338   ( \l_tmpb_dim , \l_tmpa_dim ) --
6339   ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6340   \end { tikzpicture }
6341 }

```

The command `\@@_draw_vlines:` draws all the vertical rules excepted in the blocks, in the virtual blocks (determined by a command such as `\Cdots`) and in the corners (if the key `corners` is used).

```

6342 \cs_new_protected:Npn \@@_draw_vlines:
6343 {
6344   \int_step_inline:nnn
6345   {
6346     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6347     { 2 }
6348     { 1 }
6349   }
6350   {
6351     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6352     { \c@jCol }
6353     { \int_eval:n { \c@jCol + 1 } }

```



```

6354     }
6355     {
6356         \str_if_eq:eeF { \l_@@_vlines_clist } { all }
6357         { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6358         { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6359     }
6360 }

```

The horizontal rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs of the form `{nicematrix/Rules}`.

```

6361 \cs_new_protected:Npn \@@_hline:n #1
6362 {

```

The group is for the options.

```

6363     \group_begin:
6364     \int_set_eq:NN \l_@@_end_int \c@jCol
6365     \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6366     \@@_hline_i:
6367     \group_end:
6368 }

6369 \cs_new_protected:Npn \@@_hline_i:
6370 {

```

`\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`.

```

6371     \tl_set:Nn \l_tmpa_tl { \int_use:N \l_@@_position_int }
6372     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6373     \l_tmpb_tl
6374     {

```

The boolean `\g_tmpa_bool` indicates whether the small horizontal rule will be drawn. If we find that it is in a block (a real block, created by `\Block` or a virtual block corresponding to a dotted line, created by `\Cdots`, `\Vdots`, etc.), we will set `\g_tmpa_bool` to `false` and the small horizontal rule won't be drawn.

```

6375         \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6376         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6377         { \@@_test_hline_in_block:nnnnn ##1 }

6378         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6379         { \@@_test_hline_in_block:nnnnn ##1 }

6380         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6381         { \@@_test_hline_in_stroken_block:nnnn ##1 }

6382         \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_h: }
6383         \bool_if:NTF \g_tmpa_bool
6384         {
6385             \int_if_zero:nT { \l_@@_local_start_int }

```

We keep in memory that we have a rule to draw. `\l_@@_local_start_int` will be the starting row of the rule that we will have to draw.

```

6386             { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6387         }
6388         {
6389             \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6390             {
6391                 \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6392                 \@@_hline_ii:
6393                 \int_zero:N \l_@@_local_start_int
6394             }
6395         }

```

```

6396     }
6397     \int_compare:nNtT { \l_@@_local_start_int } > { \c_zero_int }
6398     {
6399         \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6400         \@@_hline_ii:
6401     }
6402 }

6403 \cs_new_protected:Npn \@@_test_in_corner_h:
6404 {
6405     \int_compare:nNtTF { \l_tmpa_tl } = { \c_iRow + 1 }
6406     {
6407         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6408         { \bool_set_false:N \g_tmpa_bool }
6409     }
6410     {
6411         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6412         {
6413             \int_compare:nNtTF { \l_tmpa_tl } = { \c_one_int }
6414             { \bool_set_false:N \g_tmpa_bool }
6415             {
6416                 \@@_if_in_corner:nT
6417                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6418                 { \bool_set_false:N \g_tmpa_bool }
6419             }
6420         }
6421     }
6422 }

6423 \cs_new_protected:Npn \@@_hline_ii:
6424 {
6425     \tl_clear:N \l_@@_tikz_rule_tl
6426     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6427     \bool_if:NTF \l_@@_dotted_bool
6428     { \@@_hline_iv: }
6429     {
6430         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6431         { \@@_hline_iii: }
6432         { \@@_hline_v: }
6433     }
6434 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6435 \cs_new_protected:Npn \@@_hline_iii:
6436 {
6437     \pgfpicture
6438     \pgfrememberpicturepositiononpagetrue
6439     \pgf@relevantforpicturesizefalse
6440     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6441     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6442     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6443     \dim_set:Nn \l_tmpb_dim
6444     {
6445         \pgf@y
6446         - 0.5 \l_@@_rule_width_dim
6447         +
6448         ( \arrayrulewidth * \l_@@_multiplicity_int
6449           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6450     }
6451     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6452     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x

```

```

6453 \bool_lazy_all:nT
6454 {
6455   { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6456   { \cs_if_exist_p:N \CT@drsc@ }
6457   { ! \tl_if_blank_p:o \CT@drsc@ }
6458 }
6459 {
6460   \group_begin:
6461   \CT@drsc@
6462   \dim_set:Nn \l_@@_tmpd_dim
6463   {
6464     \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6465     * ( \l_@@_multiplicity_int - 1 )
6466   }
6467   \pgfpathrectanglecorners
6468   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6469   { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6470   \pgfusepathqfill
6471   \group_end:
6472 }
6473 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6474 \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6475 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6476 {
6477   \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6478   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6479   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6480 }
6481 \CT@arc@
6482 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6483 \pgfsetrectcap
6484 \pgfusepathqstroke
6485 \endpgfpicture
6486 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6487 \cs_new_protected:Npn \@@_hline_iv:
6488 {
6489   \pgfpicture
6490   \pgfrememberpicturepositiononpagetrue
6491   \pgf@relevantforpicturesizefalse
6492   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6493   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6494   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6495   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }

```

```

6496 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6497 \int_compare:nNt { \l_@@_local_start_int } = { \c_one_int }
6498 {
6499   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6500   \bool_if:NF \g_@@_delims_bool
6501   { \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.

```

6502   \tl_if_eq:NnF \g_@@_left_delim_tl (
6503     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6504   )
6505   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6506   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6507   \int_compare:nNt { \l_@@_local_end_int } = { \c_jCol }
6508   {
6509     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6510     \bool_if:NF \g_@@_delims_bool
6511     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6512     \tl_if_eq:NnF \g_@@_right_delim_tl )
6513     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6514   }
6515   \CT@arc@
6516   \@@_draw_line:
6517   \endpgfpicture
6518 }

```

The following code is for the case when the user uses the key `tikz` (in the definition of a customized rule by using the key `custom-line`).

```

6519 \cs_new_protected:Npn \@@_hline_v:
6520 {
6521   \begin { tikzpicture }

```

By default, the color defined by `\arrayrulecolor` or by `rules/color` will be used, but it's still possible to change the color by using the key `color` or, of course, the key `color` inside the key `tikz` (that is to say the key `color` provided by PGF).

```

6522   \CT@arc@
6523   \tl_if_empty:NF \l_@@_rule_color_tl
6524   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6525   \pgfrememberpicturepositiononpagetrue
6526   \pgf@relevantforpicturesizefalse
6527   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6528   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6529   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6530   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6531   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6532   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6533   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6534   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6535     ( \l_tmpa_dim , \l_tmpb_dim ) --
6536     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6537   \end { tikzpicture }
6538 }

```

The command `\@@_draw_hlines:` draws all the horizontal rules excepted in the blocks (even the virtual blocks determined by commands such as `\Cdots` and in the corners — if the key `corners` is used).

```

6539 \cs_new_protected:Npn \@@_draw_hlines:
6540 {
6541   \int_step_inline:nnn
6542     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6543   {

```

```

6544     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6545     { \c@iRow }
6546     { \int_eval:n { \c@iRow + 1 } } }
6547   }
6548   {
6549     \str_if_eq:eeF { \l_@@_hlines_clist } { all }
6550     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6551     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6552   }
6553 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6554 \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.

```

6555 \cs_set:Npn \@@_Hline_i:n #1
6556 {
6557   \peek_remove_spaces:n
6558   {
6559     \peek_meaning:NTF \Hline
6560     { \@@_Hline_ii:nn { #1 + 1 } }
6561     { \@@_Hline_iii:n { #1 } }
6562   }
6563 }
6564 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6565 \cs_set:Npn \@@_Hline_iii:n #1
6566 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6567 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6568 {
6569   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6570   \skip_vertical:N \l_@@_rule_width_dim
6571   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6572   {
6573     \@@_hline:n
6574     {
6575       multiplicity = #1 ,
6576       position = \int_eval:n { \c@iRow + 1 } ,
6577       total-width = \dim_use:N \l_@@_rule_width_dim ,
6578       #2
6579     }
6580   }
6581   \egroup
6582 }

```

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`).

```

6583 \cs_new_protected:Npn \@@_custom_line:n #1
6584 {
6585   \str_clear_new:N \l_@@_command_str
6586   \str_clear_new:N \l_@@_ccommand_str
6587   \str_clear_new:N \l_@@_letter_str
6588   \tl_clear_new:N \l_@@_other_keys_tl
6589   \keys_set:known { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl

```

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.

```

6590 \bool_lazy_all:nTF
6591 {
6592   { \str_if_empty_p:N \l_@@_letter_str }
6593   { \str_if_empty_p:N \l_@@_command_str }
6594   { \str_if_empty_p:N \l_@@_ccommand_str }
6595 }
6596 { \@@_error:n { No~letter~and~no~command } }
6597 { \@@_custom_line_i:o \l_@@_other_keys_tl }
6598 }
6599 \keys_define:nn { nicematrix / custom-line }
6600 {
6601   letter .str_set:N = \l_@@_letter_str ,
6602   letter .value_required:n = true ,
6603   command .str_set:N = \l_@@_command_str ,
6604   command .value_required:n = true ,
6605   ccommand .str_set:N = \l_@@_ccommand_str ,
6606   ccommand .value_required:n = true ,
6607 }

```

```

6608 \cs_new_protected:Npn \@@_custom_line_i:n #1
6609 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6610 \bool_set_false:N \l_@@_tikz_rule_bool
6611 \bool_set_false:N \l_@@_dotted_rule_bool
6612 \bool_set_false:N \l_@@_color_bool
6613 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6614 \bool_if:NT \l_@@_tikz_rule_bool
6615 {
6616   \IfPackageLoadedF { tikz }
6617   { \@@_error:n { tikz~in~custom~line~without~tikz } }
6618   \bool_if:NT \l_@@_color_bool
6619   { \@@_error:n { color~in~custom~line~with~tikz } }
6620 }
6621 \bool_if:NT \l_@@_dotted_rule_bool
6622 {
6623   \int_compare:nNnT { \l_@@_multiplicity_int } > { \c_one_int }
6624   { \@@_error:n { key~multiplicity~with~dotted } }
6625 }
6626 \str_if_empty:NF \l_@@_letter_str
6627 {
6628   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6629   { \@@_error:n { Several~letters } }
6630   {
6631     \tl_if_in:NoTF
6632     \c_@@_forbidden_letters_str
6633     \l_@@_letter_str
6634     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6635   }

```

During the analyse 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.

```

6636 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6637 { \@@_v_custom_line:n { #1 } }
6638 }
6639 }
6640 }

```

```

6641 \str_if_empty:NF \l_@@_command_str { \l_@@_h_custom_line:n { #1 } }
6642 \str_if_empty:NF \l_@@_ccommand_str { \l_@@_c_custom_line:n { #1 } }
6643 }
6644 \cs_generate_variant:Nn \l_@@_custom_line_i:n { o }
6645 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6646 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\l_@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{nicematrix/Rules}`). That's why the following set of keys has some keys which are no-op.

```

6647 \keys_define:nn { nicematrix / custom-line-bis }
6648 {
6649   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6650   multiplicity .initial:n = 1 ,
6651   multiplicity .value_required:n = true ,
6652   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6653   color .value_required:n = true ,
6654   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6655   tikz .value_required:n = true ,
6656   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6657   dotted .value_forbidden:n = true ,
6658   total-width .code:n = { } ,
6659   total-width .value_required:n = true ,
6660   width .code:n = { } ,
6661   width .value_required:n = true ,
6662   sep-color .code:n = { } ,
6663   sep-color .value_required:n = true ,
6664   unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6665 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6666 \bool_new:N \l_@@_dotted_rule_bool
6667 \bool_new:N \l_@@_tikz_rule_bool
6668 \bool_new:N \l_@@_color_bool

```

The following keys are used to determine the total width of the line (including the spaces on both sides of the line). The key `width` is deprecated and has been replaced by the key `total-width`.

```

6669 \keys_define:nn { nicematrix / custom-line-width }
6670 {
6671   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6672   multiplicity .initial:n = 1 ,
6673   multiplicity .value_required:n = true ,
6674   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6675   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6676   \bool_set_true:N \l_@@_total_width_bool ,
6677   total-width .value_required:n = true ,
6678   width .meta:n = { total-width = #1 } ,
6679   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6680 }

```

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 `\l_@@_hline:n` (which is in the internal `\CodeAfter`).

```

6681 \cs_new_protected:Npn \l_@@_h_custom_line:n #1
6682 {

```

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`).

```

6683   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6684   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6685 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule on only some of the columns of the array (hence the letter `c` as in `\cline`). `#1` is the whole set of keys to pass to the command `\@@_hline:n` (which is in the internal `\CodeAfter`).

```
6686 \cs_new_protected:Npn \@@_c_custom_line:n #1
6687 {
```

Here, we need an expandable command since it begins with an `\noalign`.

```
6688 \exp_args:Nc \NewExpandableDocumentCommand
6689 { \nicematrix - \l_@@_ccommand_str }
6690 { 0 { } m }
6691 {
6692 \noalign
6693 {
6694 \@@_compute_rule_width:n { #1 , ##1 }
6695 \skip_vertical:n { \l_@@_rule_width_dim }
6696 \clist_map_inline:nn
6697 { ##2 }
6698 { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6699 }
6700 }
6701 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6702 }
```

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*.

```
6703 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6704 {
6705 \tl_if_in:nnTF { #2 } { - }
6706 { \@@_cut_on_hyphen:w #2 \q_stop }
6707 { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6708 \tl_gput_right:Ne \g_@@_pre_code_after_tl
6709 {
6710 \@@_hline:n
6711 {
6712 #1 ,
6713 start = \l_tmpa_tl ,
6714 end = \l_tmpb_tl ,
6715 position = \int_eval:n { \c@iRow + 1 } ,
6716 total-width = \dim_use:N \l_@@_rule_width_dim
6717 }
6718 }
6719 }
6720 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6721 {
6722 \bool_set_false:N \l_@@_tikz_rule_bool
6723 \bool_set_false:N \l_@@_total_width_bool
6724 \bool_set_false:N \l_@@_dotted_rule_bool
6725 \keys_set_known:nn { \nicematrix / custom-line-width } { #1 }
6726 \bool_if:NF \l_@@_total_width_bool
6727 {
6728 \bool_if:NTF \l_@@_dotted_rule_bool
6729 { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6730 {
6731 \bool_if:NF \l_@@_tikz_rule_bool
6732 {
6733 \dim_set:Nn \l_@@_rule_width_dim
6734 {
6735 \arrayrulewidth * \l_@@_multiplicity_int
6736 + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6737 }
6738 }
6739 }
6740 }
6741 }
```



```

6742 \cs_new_protected:Npn \@@_v_custom_line:n #1
6743 {
6744   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6745   \tl_gput_right:Ne \g_@@_array_preamble_tl
6746     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6747   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6748     {
6749     \@@_vline:n
6750     {
6751       #1 ,
6752       position = \int_eval:n { \c@jCol + 1 } ,
6753       total-width = \dim_use:N \l_@@_rule_width_dim
6754     }
6755   }
6756   \@@_rec_preamble:n
6757 }

6758 \@@_custom_line:n
6759 { letter = : , command = hdottedline , ccommand = cdottedline, dotted }

```

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 `\l_tmpa_bool` is set to false.

```

6760 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6761 {
6762   \int_compare:nNnT { \l_tmpa_tl } > { #1 }
6763   {
6764     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6765     {
6766       \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6767       {
6768         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6769         { \bool_gset_false:N \g_tmpa_bool }
6770       }
6771     }
6772   }
6773 }

```

The same for vertical rules.

```

6774 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6775 {
6776   \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6777   {
6778     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6779     {
6780       \int_compare:nNnT { \l_tmpb_tl } > { #2 }
6781       {
6782         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6783         { \bool_gset_false:N \g_tmpa_bool }
6784       }
6785     }
6786   }
6787 }

6788 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6789 {
6790   \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6791   {
6792     \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6793     {

```

```

6794         \int_compare:nNnTF { \l_tmpa_tl } = { #1 }
6795         { \bool_gset_false:N \g_tmpa_bool }
6796         {
6797             \int_compare:nNnT { \l_tmpa_tl } = { #3 + 1 }
6798             { \bool_gset_false:N \g_tmpa_bool }
6799         }
6800     }
6801 }
6802 }
6803 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6804 {
6805     \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6806     {
6807         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6808         {
6809             \int_compare:nNnTF { \l_tmpb_tl } = { #2 }
6810             { \bool_gset_false:N \g_tmpa_bool }
6811             {
6812                 \int_compare:nNnT { \l_tmpb_tl } = { #4 + 1 }
6813                 { \bool_gset_false:N \g_tmpa_bool }
6814             }
6815         }
6816     }
6817 }

```

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.

```

6818 \cs_new_protected:Npn \@@_compute_corners:
6819 {
6820     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6821     { \@@_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`).

```

6822     \clist_clear:N \l_@@_corners_cells_clist
6823     \clist_map_inline:Nn \l_@@_corners_clist
6824     {
6825         \str_case:nnF { ##1 }
6826         {
6827             { NW }
6828             { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6829             { NE }
6830             { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6831             { SW }
6832             { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6833             { SE }
6834             { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6835         }
6836         { \@@_error:nn { bad~corner } { ##1 } }
6837     }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6838     \clist_if_empty:NF \l_@@_corners_cells_clist
6839     {

```

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.

```

6840     \tl_gput_right:Nc \g_@@_aux_tl
6841     {
6842         \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
6843         { \l_@@_corners_cells_clist }
6844     }
6845 }
6846 }

6847 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6848 {
6849     \int_step_inline:nnn { #1 } { #3 }
6850     {
6851         \int_step_inline:nnn { #2 } { #4 }
6852         { \cs_set_nopar:cpn { @@ _ block _ ##1 - ####1 } { } }
6853     }
6854 }

6855 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6856 {
6857     \cs_if_exist:cTF
6858     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6859     { \prg_return_true: }
6860     { \prg_return_false: }
6861 }

```

“Computing a corner” is determining all the empty cells (which are not in a block) that belong to that corner. These cells will be added to the sequence `\l_@@_corners_cells_clist`.

The six arguments of `\@@_compute_a_corner:nnnnnn` are as follow:

- `#1` and `#2` are the number of row and column of the cell which is actually in the corner;
- `#3` and `#4` are the steps in rows and the step in columns when moving from the corner;
- `#5` is the number of the final row when scanning the rows from the corner;
- `#6` is the number of the final column when scanning the columns from the corner.

```

6862 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6863 {

```

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.

```

6864     \bool_set_false:N \l_tmpa_bool
6865     \int_zero_new:N \l_@@_last_empty_row_int
6866     \int_set:Nn \l_@@_last_empty_row_int { #1 }
6867     \int_step_inline:nnnn { #1 } { #3 } { #5 }
6868     {
6869         \bool_lazy_or:nnTF
6870         {
6871             \cs_if_exist:pc
6872             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6873         }
6874         { \@@_if_in_block_p:nn { ##1 } { #2 } }
6875         { \bool_set_true:N \l_tmpa_bool }
6876     }

```

```

6877         \bool_if:NF \l_tmpa_bool
6878         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6879     }
6880 }

```

Now, you determine the last empty cell in the row of number 1.

```

6881 \bool_set_false:N \l_tmpa_bool
6882 \int_zero_new:N \l_@@_last_empty_column_int
6883 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6884 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6885 {
6886     \bool_lazy_or:nnTF
6887     {
6888         \cs_if_exist_p:c
6889         { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6890     }
6891     { \@@_if_in_block_p:nn { #1 } { ##1 } }
6892     { \bool_set_true:N \l_tmpa_bool }
6893     {
6894         \bool_if:NF \l_tmpa_bool
6895         { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6896     }
6897 }

```

Now, we loop over the rows.

```

6898 \int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int }
6899 {

```

We treat the row number ##1 with another loop.

```

6900     \bool_set_false:N \l_tmpa_bool
6901     \int_step_inline:nnnn { #2 } { #4 } { \l_@@_last_empty_column_int }
6902     {
6903         \bool_lazy_or:nnTF
6904         { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6905         { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6906         { \bool_set_true:N \l_tmpa_bool }
6907         {
6908             \bool_if:NF \l_tmpa_bool
6909             {
6910                 \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6911                 \clist_put_right:Nn
6912                 \l_@@_corners_cells_clist
6913                 { ##1 - #####1 }
6914                 \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6915             }
6916         }
6917     }
6918 }
6919 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6920 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6921 \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.

```
6922 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```
6923 \keys_define:nn { nicematrix / NiceMatrixBlock }
6924 {
6925   auto-columns-width .code:n =
6926   {
6927     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6928     \dim_gzero_new:N \g_@@_max_cell_width_dim
6929     \bool_set_true:N \l_@@_auto_columns_width_bool
6930   }
6931 }

6932 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6933 {
6934   \int_gincr:N \g_@@_NiceMatrixBlock_int
6935   \dim_zero:N \l_@@_columns_width_dim
6936   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6937   \bool_if:NT \l_@@_block_auto_columns_width_bool
6938   {
6939     \cs_if_exist:cT
6940     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6941     {
6942       \dim_set:Nn \l_@@_columns_width_dim
6943       {
6944         \use:c
6945         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6946       }
6947     }
6948   }
6949 }
```

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`).

```
6950 {
6951   \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.

```
6952   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6953   {
6954     \bool_if:NT \l_@@_block_auto_columns_width_bool
6955     {
6956       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6957       \iow_shipout:Ne \@mainaux
6958       {
6959         \cs_gset:cpn
6960         { @@ _ 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.

```
6961         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6962       }
6963       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6964     }
6965   }
6966   \ignorespacesafterend
6967 }
```

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).

```

6968 \cs_new_protected:Npn \@@_create_extra_nodes:
6969 {
6970   \bool_if:nTF \l_@@_medium_nodes_bool
6971   {
6972     \bool_if:NTF \l_@@_no_cell_nodes_bool
6973     { \@@_error:n { extra-nodes-with-no-cell-nodes } }
6974     {
6975       \bool_if:NTF \l_@@_large_nodes_bool
6976       \@@_create_medium_and_large_nodes:
6977       \@@_create_medium_nodes:
6978     }
6979   }
6980   {
6981     \bool_if:NT \l_@@_large_nodes_bool
6982     {
6983       \bool_if:NTF \l_@@_no_cell_nodes_bool
6984       { \@@_error:n { extra-nodes-with-no-cell-nodes } }
6985       \@@_create_large_nodes:
6986     }
6987   }
6988 }

```

We have three macros of creation of nodes: `\@@_create_medium_nodes:`, `\@@_create_large_nodes:` and `\@@_create_medium_and_large_nodes:`.

We have to compute the mathematical coordinates of the “medium nodes”. These mathematical coordinates are also used to compute the mathematical coordinates of the “large nodes”. That’s why we write a command `\@@_computations_for_medium_nodes:` to do these computations.

The command `\@@_computations_for_medium_nodes:` must be used in a `{pgfpicture}`.

For each row i , we compute two dimensions `l_@@_row_i_min_dim` and `l_@@_row_i_max_dim`. The dimension `l_@@_row_i_min_dim` is the minimal y -value of all the cells of the row i . The dimension `l_@@_row_i_max_dim` is the maximal y -value of all the cells of the row i .

Similarly, for each column j , we compute two dimensions `l_@@_column_j_min_dim` and `l_@@_column_j_max_dim`. The dimension `l_@@_column_j_min_dim` is the minimal x -value of all the cells of the column j . The dimension `l_@@_column_j_max_dim` is the maximal x -value of all the cells of the column j .

Since these dimensions will be computed as maximum or minimum, we initialize them to `\c_max_dim` or `-\c_max_dim`.

```

6989 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6990 {
6991   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6992   {
6993     \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
6994     \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
6995     \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
6996     \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
6997   }
6998   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6999   {
7000     \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7001     \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7002     \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7003     \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7004   }

```

We begin the two nested loops over the rows and the columns of the array.

```

7005 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7006 {
7007     \int_step_variable:nnNn
7008     \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.

```

7009 {
7010     \cs_if_exist:cT
7011     { pgf @ sh @ ns @ \@@_env: - \@@_i: - \@@_j: }

```

We retrieve the coordinates of the anchor south west of the (normal) node of the cell (i - j). They will be stored in `\pgf@x` and `\pgf@y`.

```

7012 {
7013     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7014     \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7015     { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7016     \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7017     {
7018         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7019         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7020     }

```

We retrieve the coordinates of the anchor north east of the (normal) node of the cell (i - j). They will be stored in `\pgf@x` and `\pgf@y`.

```

7021     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7022     \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7023     { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } { \pgf@y } }
7024     \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7025     {
7026         \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7027         { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } { \pgf@x } }
7028     }
7029 }
7030 }
7031 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7032 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7033 {
7034     \dim_compare:nNnT
7035     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7036     {
7037         \@@_qpoint:n { row - \@@_i: - base }
7038         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7039         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7040     }
7041 }
7042 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7043 {
7044     \dim_compare:nNnT
7045     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7046     {
7047         \@@_qpoint:n { col - \@@_j: }
7048         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7049         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7050     }
7051 }
7052 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7053 \cs_new_protected:Npn \@@_create_medium_nodes:
7054 {
7055   \pgfpicture
7056     \pgfrememberpicturerepositiononpagetrue
7057     \pgf@relevantforpicturesizefalse
7058     \@@_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”.

```

7059     \tl_set:Nn \l_@@_suffix_tl { -medium }
7060     \@@_create_nodes:
7061   \endpgfpicture
7062 }

```

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:`.

```

7063 \cs_new_protected:Npn \@@_create_large_nodes:
7064 {
7065   \pgfpicture
7066     \pgfrememberpicturerepositiononpagetrue
7067     \pgf@relevantforpicturesizefalse
7068     \@@_computations_for_medium_nodes:
7069     \@@_computations_for_large_nodes:
7070     \tl_set:Nn \l_@@_suffix_tl { - large }
7071     \@@_create_nodes:
7072   \endpgfpicture
7073 }

7074 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7075 {
7076   \pgfpicture
7077     \pgfrememberpicturerepositiononpagetrue
7078     \pgf@relevantforpicturesizefalse
7079     \@@_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”.

```

7080     \tl_set:Nn \l_@@_suffix_tl { - medium }
7081     \@@_create_nodes:
7082     \@@_computations_for_large_nodes:
7083     \tl_set:Nn \l_@@_suffix_tl { - large }
7084     \@@_create_nodes:
7085   \endpgfpicture
7086 }

```

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.

```

7087 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7088 {
7089   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7090   \int_set_eq:NN \l_@@_first_col_int \c_one_int

```

We have to change the values of all the dimensions `l_@@_row_i_min_dim`, `l_@@_row_i_max_dim`, `l_@@_column_j_min_dim` and `l_@@_column_j_max_dim`.

```

7091   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7092   {
7093     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }

```

¹⁵If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`


```

7094     {
7095     (
7096         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7097         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7098     )
7099     / 2
7100     }
7101     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7102     { l_@@_row _ \@@_i: _ min_dim }
7103 }
7104 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7105 {
7106     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7107     {
7108     (
7109         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7110         \dim_use:c
7111         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7112     )
7113     / 2
7114     }
7115     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7116     { l_@@_column _ \@@_j: _ max _ dim }
7117 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7118     \dim_sub:cn
7119     { l_@@_column _ 1 _ min _ dim }
7120     \l_@@_left_margin_dim
7121     \dim_add:cn
7122     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7123     \l_@@_right_margin_dim
7124 }

```

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).

```

7125 \cs_new_protected:Npn \@@_create_nodes:
7126 {
7127     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7128     {
7129         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7130         {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7131         \@@_pgf_rect_node:nnnnn
7132         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7133         { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7134         { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7135         { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7136         { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7137         \str_if_empty:NF \l_@@_name_str
7138         {
7139             \pgfnodealias
7140             { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7141             { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7142         }
7143     }
7144 }
7145 \int_step_inline:nn { \c@iRow }

```

```

7146 {
7147   \pgfnodealias
7148   { \@@_env: - ##1 - last \l_@@_suffix_tl }
7149   { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7150 }
7151 \int_step_inline:nn { \c@jCol }
7152 {
7153   \pgfnodealias
7154   { \@@_env: - last - ##1 \l_@@_suffix_tl }
7155   { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7156 }
7157 \pgfnodealias % added 2025-04-05
7158 { \@@_env: - last - last \l_@@_suffix_tl }
7159 { \@@_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 .

```

7160   \seq_map_pairwise_function:NNN
7161   \g_@@_multicolumn_cells_seq
7162   \g_@@_multicolumn_sizes_seq
7163   \@@_node_for_multicolumn:nn
7164 }

7165 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7166 {
7167   \cs_set_nopar:Npn \@@_i: { #1 }
7168   \cs_set_nopar:Npn \@@_j: { #2 }
7169 }

```

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”).

```

7170 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7171 {
7172   \@@_extract_coords_values: #1 \q_stop
7173   \@@_pgf_rect_node:nnnnn
7174   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7175   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7176   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7177   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7178   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7179   \str_if_empty:NF \l_@@_name_str
7180   {
7181     \pgfnodealias
7182     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7183     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7184   }
7185 }

```

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).

```

7186 \keys_define:nn { nicematrix / Block / FirstPass }
7187 {
7188   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7189     \bool_set_true:N \l_@@_p_block_bool ,
7190   j .value_forbidden:n = true ,
7191   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7192   l .value_forbidden:n = true ,
7193   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7194   r .value_forbidden:n = true ,
7195   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7196   c .value_forbidden:n = true ,
7197   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7198   L .value_forbidden:n = true ,
7199   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7200   R .value_forbidden:n = true ,
7201   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7202   C .value_forbidden:n = true ,
7203   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7204   t .value_forbidden:n = true ,
7205   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7206   T .value_forbidden:n = true ,
7207   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7208   b .value_forbidden:n = true ,
7209   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7210   B .value_forbidden:n = true ,
7211   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7212   m .value_forbidden:n = true ,
7213   v-center .meta:n = m ,
7214   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7215   p .value_forbidden:n = true ,
7216   color .code:n =
7217     \@@_color:n { #1 }
7218     \tl_set_rescan:Nnn
7219       \l_@@_draw_tl
7220       { \char_set_catcode_other:N ! }
7221       { #1 } ,
7222   color .value_required:n = true ,
7223   respect-arraystretch .code:n =
7224     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7225   respect-arraystretch .value_forbidden:n = true ,
7226 }

```

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.

```

7227 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7228 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7229 {

```

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).

```

7230   \tl_if_blank:nTF { #2 }
7231   { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7232   {
7233     \tl_if_in:nnTF { #2 } { - }
7234     {
7235       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7236       \@@_Block_i_czech:w \@@_Block_i:w
7237       #2 \q_stop
7238     }
7239     {
7240       \@@_error:nn { Bad~argument~for~Block } { #2 }

```

```

7241         \@@_Block_ii:nnnnn \c_one_int \c_one_int
7242     }
7243 }
7244 { #1 } { #3 } { #4 }
7245 \ignorespaces
7246 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7247 \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`.

```

7248 {
7249   \char_set_catcode_active:N -
7250   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7251 }

```

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.

```

7252 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7253 {

```

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).

```

7254   \bool_lazy_or:nnTF
7255     { \tl_if_blank_p:n { #1 } }
7256     { \str_if_eq_p:ee { * } { #1 } }
7257     { \int_set:Nn \l_tmpa_int { 100 } }
7258     { \int_set:Nn \l_tmpa_int { #1 } }
7259   \bool_lazy_or:nnTF
7260     { \tl_if_blank_p:n { #2 } }
7261     { \str_if_eq_p:ee { * } { #2 } }
7262     { \int_set:Nn \l_tmpb_int { 100 } }
7263     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7264   \int_compare:nNnTF { \l_tmpb_int } = { \c_one_int }
7265   {
7266     \tl_if_empty:NTF \l_@@_hpos_cell_tl
7267     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7268     { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7269   }
7270   { \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.

```

7271   \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7272   \tl_set:Ne \l_tmpa_tl
7273   {
7274     { \int_use:N \c@iRow }
7275     { \int_use:N \c@jCol }
7276     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7277     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7278   }

```

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).

```

7279 \bool_set_false:N \l_tmpa_bool
7280 \bool_if:NT \l_@@_amp_in_blocks_bool
\tl_if_in:nnT is slightly faster than \str_if_in:nnT.
7281 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7282 \bool_case:nF
7283 {
7284     \l_tmpa_bool { \@@_Block_vii:eenenn }
7285     \l_@@_p_block_bool { \@@_Block_vi:eenenn }

```

For the blocks mono-column, we will compose right now 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.

```

7286 \l_@@_X_bool { \@@_Block_v:eenenn }
7287 { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eenenn }
7288 { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eenenn }
7289 { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eenenn }
7290 }
7291 { \@@_Block_v:eenenn }
7292 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7293 }

```

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 now 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.

```

7294 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7295 {
7296     \int_gincr:N \g_@@_block_box_int
7297     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7298     {
7299         \tl_gput_right:Ne \g_@@_pre_code_after_tl
7300         {
7301             \@@_actually_diagbox:nnnnnn
7302             { \int_use:N \c@iRow }
7303             { \int_use:N \c@jCol }
7304             { \int_eval:n { \c@iRow + #1 - 1 } }
7305             { \int_eval:n { \c@jCol + #2 - 1 } }
7306             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7307             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7308         }
7309     }
7310     \box_gclear_new:c
7311     { 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!

```

7312 \hbox_gset:cn
7313 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7314 {

```

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`).

```

7315 \tl_if_empty:NTF \l_@@_color_tl
7316 { \int_compare:nNnT { #2 } = { \c_one_int } { \set@color } }
7317 { \@@_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`.

```

7318 \int_compare:nNnT { #1 } = { \c_one_int }
7319 {
7320 \int_if_zero:nTF { \c_iRow }
7321 {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7322 \cs_set_eq:NN \Block \@@_NullBlock:
7323 \l_@@_code_for_first_row_tl
7324 }
7325 {
7326 \int_compare:nNnT { \c_iRow } = { \l_@@_last_row_int }
7327 {
7328 \cs_set_eq:NN \Block \@@_NullBlock:
7329 \l_@@_code_for_last_row_tl
7330 }
7331 }
7332 \g_@@_row_style_tl
7333 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7334 \@@_reset_arraystretch:
7335 \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7336 #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.).

```

7337 \@@_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}`.

```

7338     \bool_if:NTF \l_@@_tabular_bool
7339     {
7340         \bool_lazy_all:nTF
7341         {
7342             { \int_compare_p:nNn { #2 } = { \c_one_int } }

```

Remind that, when the column has not a fixed width, the dimension `\l_@@_col_width_dim` has the conventional value of -1 cm.

```

7343     {
7344         ! \dim_compare_p:nNn
7345         { \l_@@_col_width_dim } < { \c_zero_dim }
7346     }
7347     { ! \g_@@_rotate_bool }
7348 }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7349     {
7350         \use:e
7351     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7352         \exp_not:N \begin { minipage }
7353         [ \str_lowercase:f \l_@@_vpos_block_str ]
7354         { \l_@@_col_width_dim }
7355         \str_case:on \l_@@_hpos_block_str
7356         { c \centering r \raggedleft l \raggedright }
7357     }
7358     #5
7359     \end { minipage }
7360 }

```

In the other cases, we use a `{tabular}`.

```

7361     {
7362         \use:e
7363     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7364         \exp_not:N \begin { tabular }
7365         [ \str_lowercase:f \l_@@_vpos_block_str ]
7366         { @ { } \l_@@_hpos_block_str @ { } }
7367     }
7368     #5
7369     \end { tabular }
7370 }
7371 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is `false`). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7372     {
7373         \c_math_toggle_token
7374         \use:e
7375     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7376         \exp_not:N \begin { array }
7377         [ \str_lowercase:f \l_@@_vpos_block_str ]
7378         { @ { } \l_@@_hpos_block_str @ { } }
7379     }
7380     #5
7381     \end { array }

```

```

7382         \c_math_toggle_token
7383     }
7384 }

```

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).

```

7385     \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.

```

7386     \int_compare:nNnT { #2 } = { \c_one_int }
7387     {
7388         \dim_gset:Nn \g_@@_blocks_wd_dim
7389         {
7390             \dim_max:nn
7391             { \g_@@_blocks_wd_dim }
7392             {
7393                 \box_wd:c
7394                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7395             }
7396         }
7397     }

```

If we are in a mono-row block we take into account the height and the depth of that block for the height and the depth of the row, excepted when the block uses explicitly an option of vertical position T or B. Remind that if the user has not used a key for the vertical position of the block, then `\l_@@_vpos_block_str` remains empty.

```

7398     \int_compare:nNnT { #1 } = { \c_one_int }
7399     {
7400         \bool_lazy_any:nT
7401         {
7402             { \str_if_empty_p:N \l_@@_vpos_block_str }
7403             { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
7404             { \str_if_eq_p:ee { \l_@@_vpos_block_str } { b } }
7405         }
7406         { \@@_adjust_blocks_ht_dp: }
7407     }
7408     \seq_gput_right:Ne \g_@@_blocks_seq
7409     {
7410         \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.

```

7411     {
7412         \exp_not:n { #3 } ,
7413         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7414     \bool_if:NT \g_@@_rotate_bool
7415     {
7416         \bool_if:NTF \g_@@_rotate_c_bool
7417         { m }
7418         {
7419             \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7420             { T }
7421         }
7422     }
7423 }
7424 {
7425     \box_use_drop:c
7426     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }

```



```

7427     }
7428   }
7429   \bool_set_false:N \g_@@_rotate_c_bool
7430 }
7431 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7432 {
7433   \dim_gset:Nn \g_@@_blocks_ht_dim
7434   {
7435     \dim_max:nn
7436     { \g_@@_blocks_ht_dim }
7437     {
7438       \box_ht:c
7439       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7440     }
7441   }
7442   \dim_gset:Nn \g_@@_blocks_dp_dim
7443   {
7444     \dim_max:nn
7445     { \g_@@_blocks_dp_dim }
7446     {
7447       \box_dp:c
7448       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7449     }
7450   }
7451 }

7452 \cs_new:Npn \@@_adjust_hpos_rotate:
7453 {
7454   \bool_if:NT \g_@@_rotate_bool
7455   {
7456     \str_set:Ne \l_@@_hpos_block_str
7457     {
7458       \bool_if:NTF \g_@@_rotate_c_bool
7459       { c }
7460       {
7461         \str_case:onF \l_@@_vpos_block_str
7462         { b l B l t r T r }
7463         {
7464           \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
7465           { r }
7466           { l }
7467         }
7468       }
7469     }
7470   }
7471 }
7472 \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*.

```

7473 \cs_new_protected:Npn \@@_rotate_box_of_block:
7474 {
7475   \box_grotate:cn
7476   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7477   { 90 }
7478   \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7479   {
7480     \vbox_gset_top:cn
7481     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7482     {
7483       \skip_vertical:n { 0.8 ex }

```

```

7484         \box_use:c
7485         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7486     }
7487 }
7488 \bool_if:NT \g_@@_rotate_c_bool
7489 {
7490     \hbox_gset:cn
7491     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7492     {
7493         \c_math_toggle_token
7494         \vcenter
7495         {
7496             \box_use:c
7497             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7498         }
7499         \c_math_toggle_token
7500     }
7501 }
7502 }

```

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 now in a box. The composition in a box will be done further, just after the construction of the array (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn`).

#1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7503 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7504 {
7505     \seq_gput_right:Ne \g_@@_blocks_seq
7506     {
7507         \l_tmpa_tl
7508         { \exp_not:n { #3 } }
7509         {
7510             \bool_if:NTF \l_@@_tabular_bool
7511             {
7512                 \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7513         \@@_reset_arraystretch:
7514         \exp_not:n
7515         {
7516             \dim_zero:N \extrarowheight
7517             #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).

```

7518         \IfPackageLoadedTF { latex-lab-testphase-table }
7519         { \tag_stop:n { table } }
7520         \use:e
7521         {
7522             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7523             { @ { } \l_@@_hpos_block_str @ { } }
7524         }
7525         #5
7526         \end { tabular }
7527     }
7528     \group_end:
7529 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7530     {
7531         \group_begin:
The following will be no-op when respect-arraystretch is in force.
7532         \@@_reset_arraystretch:
7533         \exp_not:n
7534         {
7535             \dim_zero:N \extrarowheight
7536             #4
7537             \c_math_toggle_token
7538             \use:e
7539             {
7540                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7541                 { @ { } \l_@@_hpos_block_str @ { } }
7542             }
7543             #5
7544             \end { array }
7545             \c_math_toggle_token
7546         }
7547         \group_end:
7548     }
7549 }
7550 }
7551 }
7552 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7553 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7554 {
7555     \seq_gput_right:Ne \g_@@_blocks_seq
7556     {
7557         \l_tmpa_tl
7558         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7559         { { \exp_not:n { #4 #5 } } }
7560     }
7561 }
7562 \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`.

```

7563 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7564 {
7565     \seq_gput_right:Ne \g_@@_blocks_seq
7566     {
7567         \l_tmpa_tl
7568         { \exp_not:n { #3 } }
7569         { \exp_not:n { #4 #5 } }
7570     }
7571 }
7572 \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).

```

7573 \keys_define:nn { nicematrix / Block / SecondPass }
7574 {
7575     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7576     ampersand-in-blocks .default:n = true ,
7577     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7578   tikz .code:n =
7579       \IfPackageLoadedTF { tikz }
7580       { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7581       { \@@_error:n { tikz~key~without~tikz } } ,
7582   tikz .value_required:n = true ,
7583   fill .code:n =
7584       \tl_set_rescan:Nnn
7585       \l_@@_fill_tl
7586       { \char_set_catcode_other:N ! }
7587       { #1 } ,
7588   fill .value_required:n = true ,
7589   opacity .tl_set:N = \l_@@_opacity_tl ,
7590   opacity .value_required:n = true ,
7591   draw .code:n =
7592       \tl_set_rescan:Nnn
7593       \l_@@_draw_tl
7594       { \char_set_catcode_other:N ! }
7595       { #1 } ,
7596   draw .default:n = default ,
7597   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7598   rounded-corners .default:n = 4 pt ,
7599   color .code:n =
7600       \@@_color:n { #1 }
7601       \tl_set_rescan:Nnn
7602       \l_@@_draw_tl
7603       { \char_set_catcode_other:N ! }
7604       { #1 } ,
7605   borders .clist_set:N = \l_@@_borders_clist ,
7606   borders .value_required:n = true ,
7607   hvlines .meta:n = { vlines , hlines } ,
7608   vlines .bool_set:N = \l_@@_vlines_block_bool ,
7609   vlines .default:n = true ,
7610   hlines .bool_set:N = \l_@@_hlines_block_bool ,
7611   hlines .default:n = true ,
7612   line-width .dim_set:N = \l_@@_line_width_dim ,
7613   line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7614   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7615       \bool_set_true:N \l_@@_p_block_bool ,
7616   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7617   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7618   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7619   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7620       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7621   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7622       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7623   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7624       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7625   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7626   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7627   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7628   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7629   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7630   m .value_forbidden:n = true ,
7631   v-center .meta:n = m ,
7632   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7633   p .value_forbidden:n = true ,
7634   name .tl_set:N = \l_@@_block_name_str , % .str_set:N ?
7635   name .value_required:n = true ,
7636   name .initial:n = ,
7637   respect-arraystretch .code:n =
7638       \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,

```

```

7639     respect-arraystretch .value_forbidden:n = true ,
7640     transparent .bool_set:N = \l_@@_transparent_bool ,
7641     transparent .default:n = true ,
7642     transparent .initial:n = false ,
7643     unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7644 }

```

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.

```

7645 \cs_new_protected:Npn \@@_draw_blocks:
7646 {
7647   \bool_if:NTF \c_@@_revtex_bool
7648   { \cs_set_eq:NN \ialign \@@_old_ialign: }
7649   { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7650   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7651 }
7652 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7653 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7654   \int_zero:N \l_@@_last_row_int
7655   \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”).

```

7656   \int_compare:nNnTF { #3 } > { 98 }
7657   { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7658   { \int_set:Nn \l_@@_last_row_int { #3 } }
7659   \int_compare:nNnTF { #4 } > { 98 }
7660   { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7661   { \int_set:Nn \l_@@_last_col_int { #4 } }
7662   \int_compare:nNnTF { \l_@@_last_col_int } > { \g_@@_col_total_int }
7663   {
7664     \bool_lazy_and:nnTF
7665     { \l_@@_preamble_bool }
7666     {
7667       \int_compare_p:n
7668       { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7669     }
7670     {
7671       \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7672       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7673       \@@_msg_redirect_name:nn { columns-not-used } { none }
7674     }
7675     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7676   }
7677   {
7678     \int_compare:nNnTF { \l_@@_last_row_int } > { \g_@@_row_total_int }
7679     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7680     {
7681       \@@_Block_v:nneenn
7682       { #1 }
7683       { #2 }
7684       { \int_use:N \l_@@_last_row_int }
7685       { \int_use:N \l_@@_last_col_int }
7686       { #5 }

```

```

7687         { #6 }
7688     }
7689 }
7690 }

```

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

```

7691 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7692 {

```

The group is for the keys.

```

7693     \group_begin:
7694     \int_compare:nNnT { #1 } = { #3 }
7695     { \str_set:Nn \l_@@_vpos_block_str { t } }
7696     \keys_set:nn { nicematrix / Block / SecondPass } { #5 }

```

If the content of the block contains `&`, we will have a special treatment (since the cell must be divided in several sub-cells). Remark that `\tl_if_in:nnT` is faster than `\str_if_in:nnT`.

```

7697     \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }

7698     \bool_lazy_and:nnT
7699     { \l_@@_vlines_block_bool }
7700     { ! \l_@@_ampersand_bool }
7701     {
7702         \tl_gput_right:Ne \g_nicematrix_code_after_tl
7703         {
7704             \@@_vlines_block:nnn
7705             { \exp_not:n { #5 } }
7706             { #1 - #2 }
7707             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7708         }
7709     }
7710     \bool_if:NT \l_@@_hlines_block_bool
7711     {
7712         \tl_gput_right:Ne \g_nicematrix_code_after_tl
7713         {
7714             \@@_hlines_block:nnn
7715             { \exp_not:n { #5 } }
7716             { #1 - #2 }
7717             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7718         }
7719     }
7720     \bool_if:NF \l_@@_transparent_bool
7721     {
7722         \bool_lazy_and:nnF { \l_@@_vlines_block_bool } { \l_@@_hlines_block_bool }
7723         {

```

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).

```

7724         \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7725         { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7726     }
7727 }

```

```

7728 \tl_if_empty:NF \l_@@_draw_tl
7729 {
7730     \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7731     { \@@_error:n { hlines-with-color } }
7732     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7733     {
7734         \@@_stroke_block:nnn

```

#5 are the options

```

7735         { \exp_not:n { #5 } }
7736         { #1 - #2 }
7737         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7738     }
7739     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7740     { { #1 } { #2 } { #3 } { #4 } }
7741 }
7742 \clist_if_empty:NF \l_@@_borders_clist
7743 {
7744     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7745     {
7746         \@@_stroke_borders_block:nnn
7747         { \exp_not:n { #5 } }
7748         { #1 - #2 }
7749         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7750     }
7751 }
7752 \tl_if_empty:NF \l_@@_fill_tl
7753 {
7754     \@@_add_opacity_to_fill:
7755     \tl_gput_right:Ne \g_@@_pre_code_before_tl
7756     {
7757         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7758         { #1 - #2 }
7759         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7760         { \dim_use:N \l_@@_rounded_corners_dim }
7761     }
7762 }
7763 \seq_if_empty:NF \l_@@_tikz_seq
7764 {
7765     \tl_gput_right:Ne \g_nicematrix_code_before_tl
7766     {
7767         \@@_block_tikz:nnnnn
7768         { \seq_use:Nn \l_@@_tikz_seq { , } }
7769         { #1 }
7770         { #2 }
7771         { \int_use:N \l_@@_last_row_int }
7772         { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7773     }
7774 }
7775 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7776 {
7777     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7778     {
7779         \@@_actually_diagbox:nnnnnn
7780         { #1 }
7781         { #2 }
7782         { \int_use:N \l_@@_last_row_int }
7783         { \int_use:N \l_@@_last_col_int }
7784         { \exp_not:n { ##1 } }
7785         { \exp_not:n { ##2 } }
7786     }
7787 }

```

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.

```

7788 \pgfpicture
7789 \pgfrememberpicturepositiononpagetrue
7790 \pgf@relevantforpicturesizefalse
7791 \@@_qpoint:n { row - #1 }
7792 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7793 \@@_qpoint:n { col - #2 }
7794 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7795 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7796 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7797 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7798 \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.

```

7799 \@@_pgf_rect_node:nnnnn
7800 { \@@_env: - #1 - #2 - block }
7801 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7802 \str_if_empty:NF \l_@@_block_name_str
7803 {
7804   \pgfnodealias
7805   { \@@_env: - \l_@@_block_name_str }
7806   { \@@_env: - #1 - #2 - block }
7807   \str_if_empty:NF \l_@@_name_str
7808   {
7809     \pgfnodealias
7810     { \l_@@_name_str - \l_@@_block_name_str }
7811     { \@@_env: - #1 - #2 - block }
7812   }
7813 }

```

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.

```

7814 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7815 {
7816   \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.

```

7817 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7818 {

```

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.

```

7819 \cs_if_exist:cT
7820 { pgf @ sh @ ns @ \@@_env: - #1 - #2 }

```



```

7821     {
7822         \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7823         {
7824             \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7825             \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7826         }
7827     }
7828 }

```

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.

```

7829     \dim_compare:nNnT { \l_tmpb_dim } = { \c_max_dim }
7830     {
7831         \@@_qpoint:n { col - #2 }
7832         \dim_set_eq:NN \l_tmpb_dim \pgf@x
7833     }
7834     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7835     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7836     {
7837         \cs_if_exist:cT
7838         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7839         {
7840             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7841             {
7842                 \pgfpointanchor
7843                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7844                 { east }
7845                 \dim_set:Nn \l_@@_tmpd_dim
7846                 { \dim_max:nn { \l_@@_tmpd_dim } { \pgf@x } }
7847             }
7848         }
7849     }
7850     \dim_compare:nNnT { \l_@@_tmpd_dim } = { - \c_max_dim }
7851     {
7852         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7853         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7854     }
7855     \@@_pgf_rect_node:nnnnn
7856     { \@@_env: - #1 - #2 - block - short }
7857     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7858 }

```

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.

```

7859     \bool_if:NT \l_@@_medium_nodes_bool
7860     {
7861         \@@_pgf_rect_node:nnn
7862         { \@@_env: - #1 - #2 - block - medium }
7863         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7864         {
7865             \pgfpointanchor
7866             { \@@_env:
7867               - \int_use:N \l_@@_last_row_int
7868               - \int_use:N \l_@@_last_col_int - medium
7869             }
7870             { south-east }
7871         }
7872     }
7873     \endpgfpicture
7874
7875     \bool_if:NTF \l_@@_ampersand_bool
7876     {

```

```

7877 \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7878 \int_zero_new:N \l_@@_split_int
7879 \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7880 \pgfpicture
7881 \pgfrememberpicturepositiononpagetrue
7882 \pgf@relevantforpicturesizefalse
7883
7884 \@@_qpoint:n { row - #1 }
7885 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7886 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7887 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7888 \@@_qpoint:n { col - #2 }
7889 \dim_set_eq:NN \l_tmpa_dim \pgf@x
7890 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7891 \dim_set:Nn \l_tmpb_dim
7892 { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7893 \bool_lazy_or:nnT
7894 { \l_@@_vlines_block_bool }
7895 { \str_if_eq_p:ee { \l_@@_vlines_clist } { all } }
7896 {
7897   \int_step_inline:nn { \l_@@_split_int - 1 }
7898   {
7899     \pgfpathmoveto
7900     {
7901       \pgfpoint
7902       { \l_tmpa_dim + ##1 \l_tmpb_dim }
7903       \l_@@_tmpc_dim
7904     }
7905     \pgfpathlineto
7906     {
7907       \pgfpoint
7908       { \l_tmpa_dim + ##1 \l_tmpb_dim }
7909       \l_@@_tmpd_dim
7910     }
7911     \CT@arc@
7912     \pgfsetlinewidth { 1.1 \arrayrulewidth }
7913     \pgfsetrectcap
7914     \pgfusepathqstroke
7915   }
7916 }
7917 \@@_qpoint:n { row - #1 - base }
7918 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7919 \int_step_inline:nn { \l_@@_split_int }
7920 {
7921   \group_begin:
7922   \dim_set:Nn \col@sep
7923   { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
7924   \pgftransformshift
7925   {
7926     \pgfpoint
7927     {
7928       \l_tmpa_dim + ##1 \l_tmpb_dim -
7929       \str_case:on \l_@@_hpos_block_str
7930       {
7931         l { \l_tmpb_dim + \col@sep }
7932         c { 0.5 \l_tmpb_dim }
7933         r { \col@sep }
7934       }
7935     }
7936     { \l_@@_tmpc_dim }
7937   }
7938   \pgfset { inner~sep = \c_zero_dim }
7939   \pgfnode

```

```

7940     { rectangle }
7941     {
7942       \str_case:on \l_@@_hpos_block_str
7943       {
7944         c { base }
7945         l { base~west }
7946         r { base~east }
7947       }
7948     }
7949     { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
7950   \group_end:
7951 }
7952 \endpgfpicture
7953 }

```

Now the case where there is no ampersand & in the content of the block.

```

7954 {
7955   \bool_if:NTF \l_@@_p_block_bool
7956   {

```

When the final user has used the key p, we have to compute the width.

```

7957   \pgfpicture
7958   \pgfrememberpicturepositiononpagetrue
7959   \pgf@relevantforpicturesizefalse
7960   \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7961   {
7962     \@@_qpoint:n { col - #2 }
7963     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7964     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7965   }
7966   {
7967     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7968     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7969     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7970   }
7971   \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7972 \endpgfpicture
7973 \hbox_set:Nn \l_@@_cell_box
7974 {
7975   \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
7976   { \g_tmpb_dim }
7977   \str_case:on \l_@@_hpos_block_str
7978   { c \centering r \raggedleft l \raggedright j { } }
7979   #6
7980   \end { minipage }
7981 }
7982 }
7983 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7984 \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.

```

7985 \pgfpicture
7986 \pgfrememberpicturepositiononpagetrue
7987 \pgf@relevantforpicturesizefalse
7988 \bool_lazy_any:nTF
7989 {
7990   { \str_if_empty_p:N \l_@@_vpos_block_str }
7991   { \str_if_eq_p:ee { \l_@@_vpos_block_str } { c } }
7992   { \str_if_eq_p:ee { \l_@@_vpos_block_str } { T } }
7993   { \str_if_eq_p:ee { \l_@@_vpos_block_str } { B } }
7994 }
7995 {

```

If we are in the first column, we must put the block as if it was with the key r.

```
7996 \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.

```
7997 \bool_if:nT \g_@@_last_col_found_bool
7998 {
7999 \int_compare:nNtT { #2 } = { \g_@@_col_total_int }
8000 { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8001 }
```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```
8002 \tl_set:Ne \l_tmpa_tl
8003 {
8004 \str_case:on \l_@@_vpos_block_str
8005 {
```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```
8006 { } {
8007 \str_case:on \l_@@_hpos_block_str
8008 {
8009 c { center }
8010 l { west }
8011 r { east }
8012 j { center }
8013 }
8014 }
8015 c {
8016 \str_case:on \l_@@_hpos_block_str
8017 {
8018 c { center }
8019 l { west }
8020 r { east }
8021 j { center }
8022 }
8023 }
8024 }
8025 T {
8026 \str_case:on \l_@@_hpos_block_str
8027 {
8028 c { north }
8029 l { north-west }
8030 r { north-east }
8031 j { north }
8032 }
8033 }
8034 }
8035 B {
8036 \str_case:on \l_@@_hpos_block_str
8037 {
8038 c { south }
8039 l { south-west }
8040 r { south-east }
8041 j { south }
8042 }
8043 }
8044 }
8045 }
8046 }
8047 \pgftransformshift
8048 {
8049 \pgfpointanchor
8050 {
8051 \@@_env: - #1 - #2 - block
```

```

8052         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8053     }
8054     { \l_tmpa_tl }
8055 }
8056 \pgfset { inner~sep = \c_zero_dim }
8057 \pgfnode
8058 { rectangle }
8059 { \l_tmpa_tl }
8060 { \box_use_drop:N \l_@@_cell_box } { } { }
8061 }

```

End of the case when `\l_@@_vpos_block_str` is equal to c, T or B. Now, the other cases.

```

8062 {
8063     \pgfextracty \l_tmpa_dim
8064     {
8065         \@@_qpoint:n
8066         {
8067             row - \str_if_eq:eeTF { \l_@@_vpos_block_str } { b } { #3 } { #1 }
8068             - base
8069         }
8070     }
8071     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the *x*-value of the center of the block.

```

8072     \pgfpointanchor
8073     {
8074         \@@_env: - #1 - #2 - block
8075         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8076     }
8077     {
8078         \str_case:on \l_@@_hpos_block_str
8079         {
8080             c { center }
8081             l { west }
8082             r { east }
8083             j { center }
8084         }
8085     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8086     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8087     \pgfset { inner~sep = \c_zero_dim }
8088     \pgfnode
8089     { rectangle }
8090     {
8091         \str_case:on \l_@@_hpos_block_str
8092         {
8093             c { base }
8094             l { base~west }
8095             r { base~east }
8096             j { base }
8097         }
8098     }
8099     { \box_use_drop:N \l_@@_cell_box } { } { }
8100 }
8101 \endpgfpicture
8102 }
8103 \group_end:
8104 }
8105 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }

```

For the command `\cellcolor` used within a sub-cell of a `\Block` (when the character `&` is used inside the cell).

```

8106 \cs_set_protected:Npn \@@_fill:nnnnn #1 #2 #3 #4 #5
8107 {
8108   \pgfpicture
8109   \pgfrememberpicturepositiononpagetrue
8110   \pgf@relevantforpicturesizefalse
8111   \pgfpathrectanglecorners
8112     { \pgfpoint { #2 } { #3 } }
8113     { \pgfpoint { #4 } { #5 } }
8114   \pgfsetfillcolor { #1 }
8115   \pgfusepath { fill }
8116   \endpgfpicture
8117 }

```

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).

```

8118 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8119 {
8120   \tl_if_empty:NF \l_@@_opacity_tl
8121   {
8122     \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8123       {
8124         \tl_set:Nc \l_@@_fill_tl
8125           {
8126             [ opacity = \l_@@_opacity_tl ,
8127             \tl_tail:o \l_@@_fill_tl
8128           }
8129       }
8130       {
8131         \tl_set:Nc \l_@@_fill_tl
8132           { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8133       }
8134     }
8135   }

```

The first argument of `\@@_stroke_block:nnn` is a list of options for the rectangle that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

8136 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8137 {
8138   \group_begin:
8139   \tl_clear:N \l_@@_draw_tl
8140   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8141   \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8142   \pgfpicture
8143   \pgfrememberpicturepositiononpagetrue
8144   \pgf@relevantforpicturesizefalse
8145   \tl_if_empty:NF \l_@@_draw_tl
8146   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8147     \tl_if_eq:NnTF \l_@@_draw_tl { default }
8148       { \CT@arc@ }
8149       { \@@_color:o \l_@@_draw_tl }
8150   }
8151   \pgfsetcornersarced
8152   {
8153     \pgfpoint
8154       { \l_@@_rounded_corners_dim }
8155       { \l_@@_rounded_corners_dim }
8156   }

```

```

8157 \@@_cut_on_hyphen:w #2 \q_stop
8158 \int_compare:nNnF { \l_tmpa_tl } > { \c@iRow }
8159 {
8160   \int_compare:nNnF { \l_tmpb_tl } > { \c@jCol }
8161   {
8162     \@@_qpoint:n { row - \l_tmpa_tl }
8163     \dim_set_eq:NN \l_tmpb_dim \pgf@y
8164     \@@_qpoint:n { col - \l_tmpb_tl }
8165     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8166     \@@_cut_on_hyphen:w #3 \q_stop
8167     \int_compare:nNnT { \l_tmpa_tl } > { \c@iRow }
8168     { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8169     \int_compare:nNnT { \l_tmpb_tl } > { \c@jCol }
8170     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8171     \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8172     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8173     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8174     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8175     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8176     \pgfpathrectanglecorners
8177     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8178     { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8179     \dim_compare:nNnTF { \l_@@_rounded_corners_dim } = { \c_zero_dim }
8180     { \pgfusepathqstroke }
8181     { \pgfusepath { stroke } }
8182   }
8183 }
8184 \endpgfpicture
8185 \group_end:
8186 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8187 \keys_define:nn { nicematrix / BlockStroke }
8188 {
8189   color .tl_set:N = \l_@@_draw_tl ,
8190   draw .code:n =
8191     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8192   draw .default:n = default ,
8193   line-width .dim_set:N = \l_@@_line_width_dim ,
8194   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8195   rounded-corners .default:n = 4 pt
8196 }

```

The first argument of `\@@_vlines_block:nnn` is a list of options for the rules that we will draw. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

8197 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8198 {
8199   \group_begin:
8200   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8201   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8202   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8203   \@@_cut_on_hyphen:w #2 \q_stop
8204   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8205   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8206   \@@_cut_on_hyphen:w #3 \q_stop
8207   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8208   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8209   \int_step_inline:nnn { \l_@@_tmpd_tl } { \l_tmpb_tl }
8210   {
8211     \use:e
8212     {
8213       \@@_vline:n

```

```

8214         {
8215             position = ##1 ,
8216             start = \l_@@_tmpc_tl ,
8217             end = \int_eval:n { \l_tmpa_tl - 1 } ,
8218             total-width = \dim_use:N \l_@@_line_width_dim
8219         }
8220     }
8221 }
8222 \group_end:
8223 }
8224 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8225 {
8226     \group_begin:
8227     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8228     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8229     \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8230     \@@_cut_on_hyphen:w #2 \q_stop
8231     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8232     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8233     \@@_cut_on_hyphen:w #3 \q_stop
8234     \tl_set:Nc \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8235     \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8236     \int_step_inline:nnn { \l_@@_tmpc_tl } { \l_tmpa_tl }
8237     {
8238         \use:e
8239         {
8240             \@@_hline:n
8241             {
8242                 position = ##1 ,
8243                 start = \l_@@_tmpd_tl ,
8244                 end = \int_eval:n { \l_tmpb_tl - 1 } ,
8245                 total-width = \dim_use:N \l_@@_line_width_dim
8246             }
8247         }
8248     }
8249     \group_end:
8250 }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

8251 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8252 {
8253     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8254     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8255     \dim_compare:nNnTF { \l_@@_rounded_corners_dim } > { \c_zero_dim }
8256     { \@@_error:n { borders~forbidden } }
8257     {
8258         \tl_clear_new:N \l_@@_borders_tikz_tl
8259         \keys_set:no
8260         { nicematrix / OnlyForTikzInBorders }
8261         \l_@@_borders_clist
8262         \@@_cut_on_hyphen:w #2 \q_stop
8263         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8264         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8265         \@@_cut_on_hyphen:w #3 \q_stop
8266         \tl_set:Nc \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8267         \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8268         \@@_stroke_borders_block_i:
8269     }
8270 }
8271 \hook_gput_code:nnn { begindocument } { . }

```



```

8272 {
8273   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8274   {
8275     \c_@@_pgfortikzpicture_tl
8276     \@@_stroke_borders_block_ii:
8277     \c_@@_endpgfortikzpicture_tl
8278   }
8279 }

8280 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8281 {
8282   \pgfrememberpicturepositiononpagetrue
8283   \pgf@relevantforpicturesizefalse
8284   \CT@arc@
8285   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8286   \clist_if_in:NnT \l_@@_borders_clist { right }
8287   { \@@_stroke_vertical:n \l_tmpb_tl }
8288   \clist_if_in:NnT \l_@@_borders_clist { left }
8289   { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8290   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8291   { \@@_stroke_horizontal:n \l_tmpa_tl }
8292   \clist_if_in:NnT \l_@@_borders_clist { top }
8293   { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8294 }

8295 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8296 {
8297   tikz .code:n =
8298     \cs_if_exist:NTF \tikzpicture
8299     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8300     { \@@_error:n { tikz~in~borders~without~tikz } } ,
8301   tikz .value_required:n = true ,
8302   top .code:n = ,
8303   bottom .code:n = ,
8304   left .code:n = ,
8305   right .code:n = ,
8306   unknown .code:n = \@@_error:n { bad~border }
8307 }

```

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).

```

8308 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8309 {
8310   \@@_qpoint:n \l_@@_tmpc_tl
8311   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8312   \@@_qpoint:n \l_tmpa_tl
8313   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8314   \@@_qpoint:n { #1 }
8315   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8316   {
8317     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8318     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8319     \pgfusepathqstroke
8320   }
8321   {
8322     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8323     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8324   }
8325 }

```

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).

```

8326 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8327 {

```

```

8328 \@@_qpoint:n \l_@@_tmpd_tl
8329 \clist_if_in:NnTF \l_@@_borders_clist { left }
8330 { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8331 { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8332 \@@_qpoint:n \l_tmpb_tl
8333 \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8334 \@@_qpoint:n { #1 }
8335 \tl_if_empty:NTF \l_@@_borders_tikz_tl
8336 {
8337   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8338   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8339   \pgfusepathqstroke
8340 }
8341 {
8342   \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8343   ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8344 }
8345 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

8346 \keys_define:nn { nicematrix / BlockBorders }
8347 {
8348   borders .clist_set:N = \l_@@_borders_clist ,
8349   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8350   rounded-corners .default:n = 4 pt ,
8351   line-width .dim_set:N = \l_@@_line_width_dim
8352 }

```

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.

```

8353 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8354 {
8355   \begin { tikzpicture }
8356   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because `#5` is a list of lists.

```

8357 \clist_map_inline:nn { #1 }
8358 {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8359 \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8360 \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8361 (
8362   [
8363     xshift = \dim_use:N \l_@@_offset_dim ,
8364     yshift = - \dim_use:N \l_@@_offset_dim
8365   ]
8366   #2 -| #3
8367 )
8368 rectangle
8369 (
8370   [
8371     xshift = - \dim_use:N \l_@@_offset_dim ,
8372     yshift = \dim_use:N \l_@@_offset_dim
8373   ]
8374   \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8375 ) ;

```

```

8376     }
8377   \end { tikzpicture }
8378 }
8379 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

8380 \keys_define:nn { nicematrix / SpecialOffset }
8381 { offset .dim_set:N = \l_@@_offset_dim }

```

In some circumstances, we want to nullify the command `\Block`. In order to reach that goal, we will link the command `\Block` to the following command `\@@_NullBlock`: which has the same syntax as the standard command `\Block` but which is no-op.

```

8382 \cs_new_protected:Npn \@@_NullBlock:
8383 { \@@_collect_options:n { \@@_NullBlock_i: } }
8384 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8385 { }

```

27 How to draw the dotted lines transparently

```

8386 \cs_set_protected:Npn \@@_renew_matrix:
8387 {
8388   \RenewDocumentEnvironment { pmatrix } { } {
8389     { \pNiceMatrix }
8390     { \endpNiceMatrix }
8391   \RenewDocumentEnvironment { vmatrix } { } {
8392     { \vNiceMatrix }
8393     { \endvNiceMatrix }
8394   \RenewDocumentEnvironment { Vmatrix } { } {
8395     { \VNiceMatrix }
8396     { \endVNiceMatrix }
8397   \RenewDocumentEnvironment { bmatrix } { } {
8398     { \bNiceMatrix }
8399     { \endbNiceMatrix }
8400   \RenewDocumentEnvironment { Bmatrix } { } {
8401     { \BNiceMatrix }
8402     { \endBNiceMatrix }
8403 }

```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8404 \keys_define:nn { nicematrix / Auto }
8405 {
8406   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8407   columns-type .value_required:n = true ,
8408   l .meta:n = { columns-type = l } ,
8409   r .meta:n = { columns-type = r } ,
8410   c .meta:n = { columns-type = c } ,
8411   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8412   delimiters / color .value_required:n = true ,
8413   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8414   delimiters / max-width .default:n = true ,
8415   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8416   delimiters .value_required:n = true ,
8417   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8418   rounded-corners .default:n = 4 pt
8419 }

```

```

8420 \NewDocumentCommand \AutoNiceMatrixWithDelims
8421 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8422 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8423 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8424 {

```

The group is for the protection of the keys.

```

8425 \group_begin:
8426 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8427 \use:e
8428 {
8429 \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8430 { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8431 [ \exp_not:o \l_tmpa_tl ]
8432 }
8433 \int_if_zero:nT { \l_@@_first_row_int }
8434 {
8435 \int_if_zero:nT { \l_@@_first_col_int } { & }
8436 \prg_replicate:nn { #4 - 1 } { & }
8437 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8438 }
8439 \prg_replicate:nn { #3 }
8440 {
8441 \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).

```

8442 \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8443 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8444 }
8445 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
8446 {
8447 \int_if_zero:nT { \l_@@_first_col_int } { & }
8448 \prg_replicate:nn { #4 - 1 } { & }
8449 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8450 }
8451 \end { NiceArrayWithDelims }
8452 \group_end:
8453 }

8454 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
8455 {
8456 \cs_set_protected:cpn { #1 AutoNiceMatrix }
8457 {
8458 \bool_gset_true:N \g_@@_delims_bool
8459 \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8460 \AutoNiceMatrixWithDelims { #2 } { #3 }
8461 }
8462 }

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8463 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8464 {
8465 \group_begin:
8466 \bool_gset_false:N \g_@@_delims_bool
8467 \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8468 \group_end:
8469 }

```

29 The redefinition of the command `\dotfill`

```

8470 \cs_set_eq:NN \@@_old_dotfill: \dotfill
8471 \cs_new_protected:Npn \@@_dotfill:
8472 {

```

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}`).

```

8473 \@@_old_dotfill:
8474 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8475 }

```

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`.

```

8476 \cs_new_protected:Npn \@@_dotfill_i:
8477 {
8478   \dim_compare:nNtT { \box_wd:N \l_@@_cell_box } = { \c_zero_dim }
8479   { \@@_old_dotfill: }
8480 }

```

30 The command `\diagbox`

The command `\diagbox` will be linked to `\diagbox:nn` in the environments of `nicematrix`. However, there are also redefinitions of `\diagbox` in other circumstances.

```

8481 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8482 {
8483   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8484   {
8485     \@@_actually_diagbox:nnnnnn
8486     { \int_use:N \c@iRow }
8487     { \int_use:N \c@jCol }
8488     { \int_use:N \c@iRow }
8489     { \int_use:N \c@jCol }

```

`\g_@@_row_style_tl` contains several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

The command `\@@_if_row_less:nn` is fully expandable and, thus, the instructions will be inserted in the `\g_@@_pre_code_after_tl` only if `\diagbox` is used in a row which is the scope of that chunk of instructions.

```

8490   { \g_@@_row_style_tl \exp_not:n { #1 } }
8491   { \g_@@_row_style_tl \exp_not:n { #2 } }
8492 }

```

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`.

```

8493 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8494 {
8495   { \int_use:N \c@iRow }
8496   { \int_use:N \c@jCol }
8497   { \int_use:N \c@iRow }
8498   { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8499   { }
8500 }
8501 }

```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnnn` correspond to the rectangle (=block) to slash (we recall that it's possible to use `\diagbox` in a `\Block`). The other two are the elements to draw below and above the diagonal line.

```

8502 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8503 {
8504   \pgfpicture
8505   \pgf@relevantforpicturesizefalse
8506   \pgfrememberpicturepositiononpagetrue
8507   \@@_qpoint:n { row - #1 }
8508   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8509   \@@_qpoint:n { col - #2 }
8510   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8511   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8512   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8513   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8514   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8515   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8516   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8517 }
```

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.

```

8518   \CT@arc@
8519   \pgfsetroundcap
8520   \pgfusepathqstroke
8521 }
8522 \pgfset { inner~sep = 1 pt }
8523 \pgfscope
8524 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8525 \pgfnode { rectangle } { south~west }
8526 {
8527   \begin { minipage } { 20 cm }
```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8528   \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8529   \end { minipage }
8530 }
8531 { }
8532 { }
8533 \endpgfscope
8534 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8535 \pgfnode { rectangle } { north~east }
8536 {
8537   \begin { minipage } { 20 cm }
8538   \raggedleft
8539   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8540   \end { minipage }
8541 }
8542 { }
8543 { }
8544 \endpgfpicture
8545 }
```

31 The keyword `\CodeAfter`

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment `{@@-light-syntax}` on p. 86.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
8546 \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 `\`.

```
8547 \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`.

```
8548 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8549 {
8550   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8551   \@@_CodeAfter_iv:n
8552 }
```

We catch the argument of the command `\end` (in `#1`).

```
8553 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8554 {
```

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.

```
8555   \str_if_eq:eeTF { \@currenvir } { #1 }
8556   { \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`.

```
8557   {
8558     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8559     \@@_CodeAfter_ii:n
8560   }
8561 }
```

32 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.

```
8562 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8563 {
8564   \pgfpicture
8565   \pgfrememberpicturepositiononpagetrue
8566   \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.

```
8567   \@@_qpoint:n { row - 1 }
8568   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8569   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8570   \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).

```

8571 \bool_if:nTF { #3 }
8572 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8573 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8574 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8575 {
8576   \cs_if_exist:cT
8577   { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8578   {
8579     \pgfpointanchor
8580     { \@@_env: - ##1 - #2 }
8581     { \bool_if:nTF { #3 } { west } { east } }
8582     \dim_set:Nn \l_tmpa_dim
8583     {
8584       \bool_if:nTF { #3 }
8585       { \dim_min:nn }
8586       { \dim_max:nn }
8587       \l_tmpa_dim
8588       { \pgf@x }
8589     }
8590   }
8591 }

```

Now we can put the delimiter with a node of PGF.

```

8592 \pgfset { inner~sep = \c_zero_dim }
8593 \dim_zero:N \nulldelimiterspace
8594 \pgftransformshift
8595 {
8596   \pgfpoint
8597   { \l_tmpa_dim }
8598   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8599 }
8600 \pgfnode
8601 { rectangle }
8602 { \bool_if:nTF { #3 } { east } { west } }
8603 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8604 \nullfont
8605 \c_math_toggle_token
8606 \@@_color:o \l_@@_delimiters_color_tl
8607 \bool_if:nTF { #3 } { \left #1 } { \left . }
8608 \vcenter
8609 {
8610   \nullfont
8611   \hrule \@height
8612   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8613   \@depth \c_zero_dim
8614   \@width \c_zero_dim
8615 }
8616 \bool_if:nTF { #3 } { \right . } { \right #1 }
8617 \c_math_toggle_token
8618 }
8619 { }
8620 { }
8621 \endpgfpicture
8622 }

```

33 The command `\SubMatrix`


```

8623 \keys_define:nn { nicematrix / sub-matrix }
8624 {
8625   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8626   extra-height .value_required:n = true ,
8627   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8628   left-xshift .value_required:n = true ,
8629   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8630   right-xshift .value_required:n = true ,
8631   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8632   xshift .value_required:n = true ,
8633   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8634   delimiters / color .value_required:n = true ,
8635   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8636   slim .default:n = true ,
8637   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8638   hlines .default:n = all ,
8639   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8640   vlines .default:n = all ,
8641   hvlines .meta:n = { hlines, vlines } ,
8642   hvlines .value_forbidden:n = true
8643 }
8644 \keys_define:nn { nicematrix }
8645 {
8646   SubMatrix .inherit:n = nicematrix / sub-matrix ,
8647   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8648   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8649   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8650 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8651 \keys_define:nn { nicematrix / SubMatrix }
8652 {
8653   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8654   delimiters / color .value_required:n = true ,
8655   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8656   hlines .default:n = all ,
8657   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8658   vlines .default:n = all ,
8659   hvlines .meta:n = { hlines, vlines } ,
8660   hvlines .value_forbidden:n = true ,
8661   name .code:n =
8662     \tl_if_empty:nTF { #1 }
8663     { \@@_error:n { Invalid-name } }
8664     {
8665       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8666       {
8667         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8668         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8669         {
8670           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8671           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8672         }
8673       }
8674       { \@@_error:n { Invalid-name } }
8675     } ,
8676   name .value_required:n = true ,
8677   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8678   rules .value_required:n = true ,
8679   code .tl_set:N = \l_@@_code_tl ,
8680   code .value_required:n = true ,
8681   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
8682 }

```

```

8683 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
8684 {
8685   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8686   {
8687     \SubMatrix { #1 } { #2 } { #3 } { #4 }
8688     [
8689       delimiters / color = \l_@@_delimiters_color_tl ,
8690       hlines = \l_@@_submatrix_hlines_clist ,
8691       vlines = \l_@@_submatrix_vlines_clist ,
8692       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8693       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8694       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8695       slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8696       #5
8697     ]
8698   }
8699   \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8700   \ignorespaces
8701 }
8702 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8703 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8704 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8705 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8706 {
8707   \seq_gput_right:Ne \g_@@_submatrix_seq
8708   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8709   { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8710   { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8711   { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8712   { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8713 }
8714 }

```

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).

```

8715 \NewDocumentCommand \@@_compute_i_j:nn
8716 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8717 { \@@_compute_i_j:nnnn #1 #2 }
8718 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8719 {
8720   \def \l_@@_first_i_tl { #1 }
8721   \def \l_@@_first_j_tl { #2 }
8722   \def \l_@@_last_i_tl { #3 }
8723   \def \l_@@_last_j_tl { #4 }
8724   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8725     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8726   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8727     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8728   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8729     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8730   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8731     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8732 }

```

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`.

```

8733 \hook_gput_code:nnn { begindocument } { . }
8734 {
8735   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
8736   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
8737     { \@@_sub_matrix:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
8738 }
8739 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
8740 {
8741   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8742   \@@_compute_i_j:nn { #2 } { #3 }
8743   \int_compare:nNt { \l_@@_first_i_tl } = { \l_@@_last_i_tl }
8744   { \def \arraystretch { 1 } }
8745   \bool_lazy_or:nnTF
8746     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
8747     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
8748     { \@@_error:nn { Construct~too-large } { \SubMatrix } }
8749   {
8750     \str_clear_new:N \l_@@_submatrix_name_str
8751     \keys_set:nn { nicematrix / SubMatrix } { #5 }
8752     \pgfpicture
8753     \pgfrememberpicturepositiononpagetrue
8754     \pgf@relevantforpicturesizefalse
8755     \pgfset { inner~sep = \c_zero_dim }
8756     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8757     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curryfication.

```

8758   \bool_if:NTF \l_@@_submatrix_slim_bool
8759   { \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl } }
8760   { \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int } }
8761   {
8762     \cs_if_exist:cT
8763     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8764     {
8765       \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8766       \dim_compare:nNt { \pgf@x } < { \l_@@_x_initial_dim }
8767       { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8768     }
8769     \cs_if_exist:cT
8770     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8771     {
8772       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8773       \dim_compare:nNt { \pgf@x } > { \l_@@_x_final_dim }
8774       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8775     }
8776   }
8777   \dim_compare:nNtTF { \l_@@_x_initial_dim } = { \c_max_dim }
8778   { \@@_error:nn { Impossible~delimiter } { left } }
8779   {
8780     \dim_compare:nNtTF { \l_@@_x_final_dim } = { - \c_max_dim }

```

```

8781         { \@@_error:nn { Impossible-delimiter } { right } }
8782         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8783     }
8784     \endpgfpicture
8785 }
8786 \group_end:
8787 \ignorespaces
8788 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8789 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8790 {
8791     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8792     \dim_set:Nn \l_@@_y_initial_dim
8793     {
8794         \fp_to_dim:n
8795         {
8796             \pgf@y
8797             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8798         }
8799     }
8800     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8801     \dim_set:Nn \l_@@_y_final_dim
8802     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8803     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
8804     {
8805         \cs_if_exist:cT
8806         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8807         {
8808             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8809             \dim_set:Nn \l_@@_y_initial_dim
8810             { \dim_max:nn { \l_@@_y_initial_dim } { \pgf@y } }
8811         }
8812         \cs_if_exist:cT
8813         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8814         {
8815             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8816             \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
8817             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8818         }
8819     }
8820     \dim_set:Nn \l_tmpa_dim
8821     {
8822         \l_@@_y_initial_dim - \l_@@_y_final_dim +
8823         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8824     }
8825     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the \SubMatrix.

```

8826     \group_begin:
8827     \pgfsetlinewidth { 1.1 \arrayrulewidth }
8828     \@@_set_CTarc:o \l_@@_rules_color_tl
8829     \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`.

```

8830     \seq_map_inline:Nn \g_@@_cols_vlism_seq
8831     {
8832         \int_compare:nNnT { \l_@@_first_j_tl } < { ##1 }
8833         {
8834             \int_compare:nNnT
8835             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }

```

```
8836         {
```

First, we extract the value of the abscissa of the rule we have to draw.

```
8837             \@_qpoint:n { col - ##1 }
8838             \pgfpathmoveto { \pgfpoint \pgf@x \l_@_y_initial_dim }
8839             \pgfpathlineto { \pgfpoint \pgf@x \l_@_y_final_dim }
8840             \pgfusepathqstroke
8841         }
8842     }
8843 }
```

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.

```
8844 \str_if_eq:eeTF { \l_@_submatrix_vlines_clist } { all }
8845 { \int_step_inline:nn { \l_@_last_j_tl - \l_@_first_j_tl } }
8846 { \clist_map_inline:Nn \l_@_submatrix_vlines_clist }
8847 {
8848     \bool_lazy_and:nnTF
8849     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
8850     {
8851         \int_compare_p:nNn
8852         { ##1 } < { \l_@_last_j_tl - \l_@_first_j_tl + 1 } }
8853     {
8854         \@_qpoint:n { col - \int_eval:n { ##1 + \l_@_first_j_tl } }
8855         \pgfpathmoveto { \pgfpoint \pgf@x \l_@_y_initial_dim }
8856         \pgfpathlineto { \pgfpoint \pgf@x \l_@_y_final_dim }
8857         \pgfusepathqstroke
8858     }
8859     { \@_error:nmn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8860 }
```

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.

```
8861 \str_if_eq:eeTF { \l_@_submatrix_hlines_clist } { all }
8862 { \int_step_inline:nn { \l_@_last_i_tl - \l_@_first_i_tl } }
8863 { \clist_map_inline:Nn \l_@_submatrix_hlines_clist }
8864 {
8865     \bool_lazy_and:nnTF
8866     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
8867     {
8868         \int_compare_p:nNn
8869         { ##1 } < { \l_@_last_i_tl - \l_@_first_i_tl + 1 } }
8870     {
8871         \@_qpoint:n { row - \int_eval:n { ##1 + \l_@_first_i_tl } }
```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```
8872     \group_begin:
```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```
8873     \dim_set:Nn \l_tmpa_dim
8874     { \l_@_x_initial_dim - \l_@_submatrix_left_xshift_dim }
8875     \str_case:nn { #1 }
8876     {
8877         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8878         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8879         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8880     }
8881     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```
8882     \dim_set:Nn \l_tmpb_dim
8883     { \l_@_x_final_dim + \l_@_submatrix_right_xshift_dim }
8884     \str_case:nn { #2 }
8885     {
8886         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
```

```

8887         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8888         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8889     }
8890     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8891     \pgfusepathqstroke
8892     \group_end:
8893 }
8894 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8895 }

```

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).

```

8896 \str_if_empty:NF \l_@@_submatrix_name_str
8897 {
8898     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8899     \l_@@_x_initial_dim \l_@@_y_initial_dim
8900     \l_@@_x_final_dim \l_@@_y_final_dim
8901 }
8902 \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`.

```

8903 \begin { pgfscope }
8904 \pgftransformshift
8905 {
8906     \pgfpoint
8907     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8908     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8909 }
8910 \str_if_empty:NTF \l_@@_submatrix_name_str
8911 { \@@_node_left:nn #1 { } }
8912 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8913 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8914 \pgftransformshift
8915 {
8916     \pgfpoint
8917     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8918     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8919 }
8920 \str_if_empty:NTF \l_@@_submatrix_name_str
8921 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8922 {
8923     \@@_node_right:nnnn #2
8924     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8925 }

```

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`.

```

8926 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8927 \flag_clear_new:N \l_@@_code_flag
8928 \l_@@_code_tl
8929 }

```

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`.

```

8930 \cs_set_eq:NN \@@_old_pgfpointanchor: \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by curryfication.

```
8931 \cs_new:Npn \@@_pgfpointanchor:n #1
8932 { \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`).

```
8933 \cs_new:Npn \@@_pgfpointanchor_i:n #1
8934 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }
8935 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
8936 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
8937 \str_if_empty:nTF { #1 }
```

First, when the name of the name begins with `\tikz@pp@name`.

```
8938 { \@@_pgfpointanchor_iv:w #2 }
```

And now, when there is no `\tikz@pp@name`.

```
8939 { \@@_pgfpointanchor_ii:n { #1 } }
8940 }
```

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`).

```
8941 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
8942 { \@@_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`.

```
8943 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

8944 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
8945 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
8946 \str_if_empty:nTF { #2 }
```

First the case where the argument does *not* contain an hyphen.

```
8947 { \@@_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`).

```
8948 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8949 }
```

The following function is for the case when the name contains an hyphen.

```
8950 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8951 {
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
8952 \@@_env:
8953 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8954 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8955 }
```

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.

```

8956 \tl_const:Nn \c_@@_integers_alist_tl
8957 {
8958   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8959   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8960   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8961   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8962 }

8963 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
8964 {

```

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`.

```

8965   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8966   {
8967     \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

8968   \@@_env: -
8969   \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8970     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8971     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8972   }
8973   {
8974     \str_if_eq:eeTF { #1 } { last }
8975     {
8976       \flag_raise:N \l_@@_code_flag
8977       \@@_env: -
8978       \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8979         { \int_eval:n { \l_@@_last_i_tl + 1 } }
8980         { \int_eval:n { \l_@@_last_j_tl + 1 } }
8981     }
8982     { #1 }
8983   }
8984 }

```

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`).

```

8985 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8986 {
8987   \pgfnode
8988     { rectangle }
8989     { east }
8990     {
8991       \nullfont
8992       \c_math_toggle_token
8993       \@@_color:o \l_@@_delimiters_color_tl
8994       \left #1
8995       \vcenter
8996       {
8997         \nullfont
8998         \hrule \@height \l_tmpa_dim
8999         \@depth \c_zero_dim

```



```

9000             \@width \c_zero_dim
9001         }
9002     \right .
9003     \c_math_toggle_token
9004 }
9005 { #2 }
9006 { }
9007 }

```

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.

```

9008 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9009 {
9010     \pgfnode
9011     { rectangle }
9012     { west }
9013     {
9014         \nullfont
9015         \c_math_toggle_token
9016         \colorlet { current-color } { . }
9017         \@@_color:o \l_@@_delimiters_color_tl
9018         \left .
9019         \vcenter
9020         {
9021             \nullfont
9022             \hrule \@height \l_tmpa_dim
9023                 \@depth \c_zero_dim
9024                 \@width \c_zero_dim
9025         }
9026         \right #1
9027         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9028         ^ { \color { current-color } \smash { #4 } }
9029         \c_math_toggle_token
9030     }
9031     { #2 }
9032     { }
9033 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9034 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9035 {
9036     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9037     \ignorespaces
9038 }
9039 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9040 {
9041     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9042     \ignorespaces
9043 }
9044 \keys_define:nn { nicematrix / Brace }
9045 {
9046     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9047     left-shorten .default:n = true ,
9048     left-shorten .value_forbidden:n = true ,

```

```

9049   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9050   right-shorten .default:n = true ,
9051   right-shorten .value_forbidden:n = true ,
9052   shorten .meta:n = { left-shorten , right-shorten } ,
9053   shorten .value_forbidden:n = true ,
9054   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9055   yshift .value_required:n = true ,
9056   yshift .initial:n = \c_zero_dim ,
9057   color .tl_set:N = \l_tmpa_tl ,
9058   color .value_required:n = true ,
9059   unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
9060 }

```

#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.

```

9061 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9062 {
9063   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9064   \@@_compute_i_j:nn { #1 } { #2 }
9065   \bool_lazy_or:nnTF
9066     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9067     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9068   {
9069     \str_if_eq:eeTF { #5 } { under }
9070     { \@@_error:nn { Construct~too~large } { \UnderBrace } }
9071     { \@@_error:nn { Construct~too~large } { \OverBrace } }
9072   }
9073   {
9074     \tl_clear:N \l_tmpa_tl
9075     \keys_set:nn { nicematrix / Brace } { #4 }
9076     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9077     \pgfpicture
9078     \pgfrememberpicturepositiononpagetrue
9079     \pgf@relevantforpicturesizefalse
9080     \bool_if:NT \l_@@_brace_left_shorten_bool
9081     {
9082       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9083       \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9084       {
9085         \cs_if_exist:cT
9086           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9087         {
9088           \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9089
9090           \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9091             { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9092         }
9093       }
9094     }
9095     \bool_lazy_or:nnT
9096       { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9097       { \dim_compare_p:nNn { \l_@@_x_initial_dim } = { \c_max_dim } }
9098     {
9099       \@@_qpoint:n { col - \l_@@_first_j_tl }
9100       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9101     }
9102     \bool_if:NT \l_@@_brace_right_shorten_bool
9103     {
9104       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9105       \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9106       {

```

```

9107         \cs_if_exist:cT
9108         { pgf @ sh @ ns @ \l_@@_env: - ##1 - \l_@@_last_j_tl }
9109         {
9110             \pgfpointanchor { \l_@@_env: - ##1 - \l_@@_last_j_tl } { east }
9111             \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
9112             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9113         }
9114     }
9115 }
9116 \bool_lazy_or:nnT
9117 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9118 { \dim_compare_p:nNn { \l_@@_x_final_dim } = { - \c_max_dim } }
9119 {
9120     \l_@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9121     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9122 }
9123 \pgfset { inner~sep = \c_zero_dim }
9124 \str_if_eq:eeTF { #5 } { under }
9125 { \l_@@_underbrace_i:n { #3 } }
9126 { \l_@@_overbrace_i:n { #3 } }
9127 \endpgfpicture
9128 }
9129 \group_end:
9130 }

```

The argument is the text to put above the brace.

```

9131 \cs_new_protected:Npn \l_@@_overbrace_i:n #1
9132 {
9133     \l_@@_qpoint:n { row - \l_@@_first_i_tl }
9134     \pgftransformshift
9135     {
9136         \pgfpoint
9137         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9138         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9139     }
9140     \pgfnode
9141     { rectangle }
9142     { south }
9143     {
9144         \vtop
9145         {
9146             \group_begin:
9147             \everycr { }
9148             \halign
9149             {
9150                 \hfil ## \hfil \crcr
9151                 \bool_if:NTF \l_@@_tabular_bool
9152                 { \begin { tabular } { c } #1 \end { tabular } }
9153                 { $ \begin { array } { c } #1 \end { array } $ }
9154                 \cr
9155                 \c_math_toggle_token
9156                 \overbrace
9157                 {
9158                     \hbox_to_wd:nn
9159                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9160                     { }
9161                 }
9162                 \c_math_toggle_token
9163                 \cr
9164             }
9165             \group_end:
9166         }
9167     }
9168 }

```

```

9169     { }
9170 }

```

The argument is the text to put under the brace.

```

9171 \cs_new_protected:Npn \@@_underbrace_i:n #1
9172 {
9173   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9174   \pgftransformshift
9175   {
9176     \pgfpoint
9177     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9178     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9179   }
9180   \pgfnode
9181   { rectangle }
9182   { north }
9183   {
9184     \group_begin:
9185     \everycr { }
9186     \vbox
9187     {
9188       \halign
9189       {
9190         \hfil ## \hfil \crcr
9191         \c_math_toggle_token
9192         \underbrace
9193         {
9194           \hbox_to_wd:nn
9195           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9196           { }
9197         }
9198         \c_math_toggle_token
9199         \cr
9200         \bool_if:NTF \l_@@_tabular_bool
9201         { \begin { tabular } { c } #1 \end { tabular } }
9202         { $ \begin { array } { c } #1 \end { array } $ }
9203         \cr
9204       }
9205     }
9206     \group_end:
9207   }
9208   { }
9209   { }
9210 }

```

35 The commands HBrace et VBrace

```

9211 \hook_gput_code:nnn { begindocument } { . }
9212 {
9213   \cs_if_exist:cT { tikz@library@decorations.pathreplacing@loaded }
9214   {
9215     \tikzset
9216     {
9217       nicematrix / brace / .style =
9218       {
9219         decoration = { brace , raise = -0.15 em } ,
9220         decorate ,
9221       } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9222         nicematrix / mirrored-brace / .style =
9223         {
9224             nicematrix / brace ,
9225             decoration = mirror ,
9226         }
9227     }
9228 }
9229 }
```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9230 \keys_define:nn { nicematrix / Hbrace }
9231 {
9232     color .code:n = ,
9233     horizontal-label .code:n = ,
9234     horizontal-labels .code:n = ,
9235     shorten .code:n = ,
9236     shorten-start .code:n = ,
9237     shorten-end .code:n = ,
9238     unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9239 }
```

Here we need an “fully expandable” command.

```

9240 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9241 {
9242     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9243     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9244     { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9245 }
```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

9246 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9247 {
9248     \int_compare:nNnTF { \c@iRow } < { 2 }
9249     {
```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9250     \str_if_eq:nnTF { #2 } { * }
9251     {
9252         \NiceMatrixOptions { nullify-dots }
9253         \Ldots
9254         [
9255             line-style = nicematrix / brace ,
9256             #1 ,
9257             up =
9258             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9259         ]
9260     }
9261     {
9262         \Hdotsfor
9263         [
9264             line-style = nicematrix / brace ,
9265             #1 ,
9266             up =
9267             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9268         ]
9269         { #2 }
9270     }
9271 }
9272 {
9273     \str_if_eq:nnTF { #2 } { * }
```

```

9274 {
9275   \NiceMatrixOptions { nullify-dots }
9276   \Ldots
9277   [
9278     line-style = nicematrix / mirrored-brace ,
9279     #1 ,
9280     down =
9281       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9282   ]
9283 }
9284 {
9285   \Hdotsfor
9286   [
9287     line-style = nicematrix / mirrored-brace ,
9288     #1 ,
9289     down =
9290       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9291   ]
9292   { #2 }
9293 }
9294 }
9295 \keys_set:nn { nicematrix / Hbrace } { #1 }
9296 }

9297 \NewDocumentCommand { \@@_Vbrace } { 0 { } m m }
9298 {
9299   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9300   { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9301   { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9302 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

9303 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
9304 {
9305   \int_compare:nNnTF { \c@jCol } < { 2 }
9306   {
9307     \str_if_eq:nnTF { #2 } { * }
9308     {
9309       \NiceMatrixOptions { nullify-dots }
9310       \Vdots
9311       [
9312         Vbrace ,
9313         line-style = nicematrix / mirrored-brace ,
9314         #1 ,
9315         down =
9316           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9317       ]
9318     }
9319     {
9320       \Vdotsfor
9321       [
9322         Vbrace ,
9323         line-style = nicematrix / mirrored-brace ,
9324         #1 ,
9325         down =
9326           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9327       ]
9328       { #2 }
9329     }
9330   }
9331   {
9332     \str_if_eq:nnTF { #2 } { * }
9333     {

```

```

9334 \NiceMatrixOptions { nullify-dots }
9335 \Vdots
9336 [
9337   Vbrace ,
9338   line-style = nicematrix / brace ,
9339   #1 ,
9340   up =
9341   \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9342 ]
9343 }
9344 {
9345   \Vdotsfor
9346   [
9347     Vbrace ,
9348     line-style = nicematrix / brace ,
9349     #1 ,
9350     up =
9351     \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9352   ]
9353   { #2 }
9354 }
9355 }
9356 \keys_set:nn { nicematrix / Hbrace } { #1 }
9357 }

```

36 The command TikzEveryCell

```

9358 \bool_new:N \l_@@_not_empty_bool
9359 \bool_new:N \l_@@_empty_bool
9360
9361 \keys_define:nn { nicematrix / TikzEveryCell }
9362 {
9363   not-empty .code:n =
9364     \bool_lazy_or:nnTF
9365     { \l_@@_in_code_after_bool }
9366     { \g_@@_create_cell_nodes_bool }
9367     { \bool_set_true:N \l_@@_not_empty_bool }
9368     { \@@_error:n { detection-of-empty-cells } } ,
9369   not-empty .value_forbidden:n = true ,
9370   empty .code:n =
9371     \bool_lazy_or:nnTF
9372     { \l_@@_in_code_after_bool }
9373     { \g_@@_create_cell_nodes_bool }
9374     { \bool_set_true:N \l_@@_empty_bool }
9375     { \@@_error:n { detection-of-empty-cells } } ,
9376   empty .value_forbidden:n = true ,
9377   unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
9378 }
9379
9380
9381 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9382 {
9383   \IfPackageLoadedTF { tikz }
9384   {
9385     \group_begin:
9386     \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.

```

9387   \tl_set:Nn \l_tmpa_tl { { #2 } }
9388   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9389   { \@@_for_a_block:nnnnn ##1 }

```

```

9390     \@@_all_the_cells:
9391     \group_end:
9392 }
9393 { \@@_error:n { TikzEveryCell~without~tikz } }
9394 }
9395
9396
9397 \cs_new_protected:Nn \@@_all_the_cells:
9398 {
9399   \int_step_inline:nn \c@iRow
9400   {
9401     \int_step_inline:nn \c@jCol
9402     {
9403       \cs_if_exist:cF { cell - ##1 - #####1 }
9404       {
9405         \clist_if_in:NcF \l_@@_corners_cells_clist
9406         { ##1 - #####1 }
9407         {
9408           \bool_set_false:N \l_tmpa_bool
9409           \cs_if_exist:cTF
9410           { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
9411           {
9412             \bool_if:NF \l_@@_empty_bool
9413             { \bool_set_true:N \l_tmpa_bool }
9414           }
9415           {
9416             \bool_if:NF \l_@@_not_empty_bool
9417             { \bool_set_true:N \l_tmpa_bool }
9418           }
9419           \bool_if:NT \l_tmpa_bool
9420           {
9421             \@@_block_tikz:nnnn
9422             \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
9423           }
9424         }
9425       }
9426     }
9427   }
9428 }
9429
9430 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9431 {
9432   \bool_if:NF \l_@@_empty_bool
9433   {
9434     \@@_block_tikz:nnnnn
9435     \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9436   }
9437   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9438 }
9439
9440 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9441 {
9442   \int_step_inline:nnn { #1 } { #3 }
9443   {
9444     \int_step_inline:nnn { #2 } { #4 }
9445     { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9446   }
9447 }

```

37 The command \ShowCellNames

```

9448 \NewDocumentCommand \@@_ShowCellNames { }

```



```

9449 {
9450   \bool_if:NT \l_@@_in_code_after_bool
9451   {
9452     \pgfpicture
9453     \pgfrememberpicturepositiononpagetrue
9454     \pgf@relevantforpicturesizefalse
9455     \pgfpathrectanglecorners
9456     { \@@_qpoint:n { 1 } }
9457     {
9458       \@@_qpoint:n
9459       { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
9460     }
9461     \pgfsetfillopacity { 0.75 }
9462     \pgfsetfillcolor { white }
9463     \pgfusepathqfill
9464     \endpgfpicture
9465   }
9466   \dim_gzero_new:N \g_@@_tmpc_dim
9467   \dim_gzero_new:N \g_@@_tmpd_dim
9468   \dim_gzero_new:N \g_@@_tmpe_dim
9469   \int_step_inline:nn { \c@iRow }
9470   {
9471     \bool_if:NTF \l_@@_in_code_after_bool
9472     {
9473       \pgfpicture
9474       \pgfrememberpicturepositiononpagetrue
9475       \pgf@relevantforpicturesizefalse
9476     }
9477     { \begin { pgfpicture } }
9478     \@@_qpoint:n { row - ##1 }
9479     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9480     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9481     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9482     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9483     \bool_if:NTF \l_@@_in_code_after_bool
9484     { \endpgfpicture }
9485     { \end { pgfpicture } }
9486     \int_step_inline:nn { \c@jCol }
9487     {
9488       \hbox_set:Nn \l_tmpa_box
9489       {
9490         \normalfont \Large \sffamily \bfseries
9491         \bool_if:NTF \l_@@_in_code_after_bool
9492         { \color { red } }
9493         { \color { red ! 50 } }
9494         ##1 - ####1
9495       }
9496       \bool_if:NTF \l_@@_in_code_after_bool
9497       {
9498         \pgfpicture
9499         \pgfrememberpicturepositiononpagetrue
9500         \pgf@relevantforpicturesizefalse
9501       }
9502       { \begin { pgfpicture } }
9503       \@@_qpoint:n { col - ####1 }
9504       \dim_set_eq:NN \g_@@_tmpc_dim \pgf@x
9505       \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9506       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9507       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9508       \bool_if:NTF \l_@@_in_code_after_bool
9509       { \endpgfpicture }
9510       { \end { pgfpicture } }
9511       \fp_set:Nn \l_tmpa_fp

```

```

9512         {
9513             \fp_min:nn
9514             {
9515                 \fp_min:nn
9516                 { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9517                 { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9518             }
9519             { 1.0 }
9520         }
9521     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9522     \pgfpicture
9523     \pgfrememberpicturepositiononpagetrue
9524     \pgf@relevantforpicturesizefalse
9525     \pgftransformshift
9526     {
9527         \pgfpoint
9528         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9529         { \dim_use:N \g_tmpa_dim }
9530     }
9531     \pgfnode
9532     { rectangle }
9533     { center }
9534     { \box_use:N \l_tmpa_box }
9535     { }
9536     { }
9537     \endpgfpicture
9538 }
9539 }
9540 }

```

38 We process the options at package loading

We process the options when the package is loaded (with `\usepackage`) but we recommend to use `\NiceMatrixOptions` instead.

We must process these options after the definition of the environment `{NiceMatrix}` because the option `renew-matrix` executes the code `\cs_set_eq:NN \env@matrix \NiceMatrix`.

Of course, the command `\NiceMatrix` must be defined before such an instruction is executed.

The boolean `\g_@@_footnotehyper_bool` will indicate if the option `footnotehyper` is used.

```

9541 \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.

```

9542 \bool_new:N \g_@@_footnote_bool

9543 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9544 {
9545     You~have~used~the~key~' \l_keys_key_str '~when~loading~nicematrix~
9546     but~that~key~is~unknown. \\
9547     It~will~be~ignored. \\
9548     For~a~list~of~the~available~keys,~type~H~<return>.
9549 }
9550 {
9551     The~available~keys~are~(in~alphabetic~order):~
9552     footnote,~
9553     footnotehyper,~
9554     messages~for~Overleaf,~
9555     renew~dots~and~
9556     renew~matrix.
9557 }

```

```

9558 \keys_define:nn { nicematrix }
9559 {
9560   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9561   renew-dots .value_forbidden:n = true ,
9562   renew-matrix .code:n = \@@_renew_matrix: ,
9563   renew-matrix .value_forbidden:n = true ,
9564   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9565   footnote .bool_set:N = \g_@@_footnote_bool ,
9566   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9567   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9568 }
9569 \ProcessKeyOptions

9570 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9571 {
9572   You~can't~use~the~option~'footnote'~because~the~package~
9573   footnotehyper~has~already~been~loaded.~
9574   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9575   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9576   of~the~package~footnotehyper.\\
9577   The~package~footnote~won't~be~loaded.
9578 }

9579 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9580 {
9581   You~can't~use~the~option~'footnotehyper'~because~the~package~
9582   footnote~has~already~been~loaded.~
9583   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9584   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9585   of~the~package~footnote.\\
9586   The~package~footnotehyper~won't~be~loaded.
9587 }

9588 \bool_if:NT \g_@@_footnote_bool
9589 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9590 \IfClassLoadedTF { beamer }
9591 { \bool_set_false:N \g_@@_footnote_bool }
9592 {
9593   \IfPackageLoadedTF { footnotehyper }
9594   { \@@_error:n { footnote~with~footnotehyper~package } }
9595   { \usepackage { footnote } }
9596 }
9597 }

9598 \bool_if:NT \g_@@_footnotehyper_bool
9599 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9600 \IfClassLoadedTF { beamer }
9601 { \bool_set_false:N \g_@@_footnote_bool }
9602 {
9603   \IfPackageLoadedTF { footnote }
9604   { \@@_error:n { footnotehyper~with~footnote~package } }
9605   { \usepackage { footnotehyper } }
9606 }
9607 \bool_set_true:N \g_@@_footnote_bool
9608 }

```

The flag `\g_@@_footnote_bool` is raised and so, we will only have to test `\g_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

39 About the package underscore

If the user loads the package underscore, it must be loaded *before* the package nicematrix. If it is loaded after, we raise an error.

```
9609 \bool_new:N \l_@@_underscore_loaded_bool
9610 \IfPackageLoadedT { underscore }
9611 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9612 \hook_gput_code:nnn { begindocument } { . }
9613 {
9614   \bool_if:NF \l_@@_underscore_loaded_bool
9615   {
9616     \IfPackageLoadedT { underscore }
9617     { \@@_error:n { underscore-after-nicematrix } }
9618   }
9619 }
```

40 Error messages of the package

```
9620 \str_const:Ne \c_@@_available_keys_str
9621 {
9622   \bool_if:nTF { ! \g_@@_messages_for_Overleaf_bool }
9623   { For~a~list~of~the~available~keys,~type~H~<return>. }
9624   { }
9625 }
9626 \seq_new:N \g_@@_types_of_matrix_seq
9627 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9628 {
9629   NiceMatrix ,
9630   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9631 }
9632 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9633 { \tl_to_str:n { #1 } }
```

If the user uses too much columns, the command `\@@_error_too_much_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`.

```
9634 \cs_new_protected:Npn \@@_error_too_much_cols:
9635 {
9636   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9637   { \@@_fatal:nn { too-much-cols-for-array } }
9638   \int_compare:nNnT { \l_@@_last_col_int } = { -2 }
9639   { \@@_fatal:n { too-much-cols-for-matrix } }
9640   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
9641   { \@@_fatal:n { too-much-cols-for-matrix } }
9642   \bool_if:NF \l_@@_last_col_without_value_bool
9643   { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9644 }
```

The following command must *not* be protected since it's used in an error message.

```
9645 \cs_new:Npn \@@_message_hdotsfor:
9646 {
9647   \tl_if_empty:oF \g_@@_HVDotsfor_lines_tl
9648   { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
9649     \token_to_str:N \Hbrace \ is~incorrect. }
9650 }
```

```

9651 \cs_new_protected:Npn \@@_Hline_in_cell:
9652 { \@@_fatal:n { Misuse-of-Hline } }

9653 \@@_msg_new:nn { Misuse-of-Hline }
9654 {
9655     Misuse-of-Hline. \\
9656     \token_to_str:N \Hline\ must-be-used-only-at-the-beginning-of-a-row.\\
9657     That-error-is-fatal.
9658 }

9659 \@@_msg_new:nn { hvlines,~rounded-corners-and-corners }
9660 {
9661     Incompatible-options.\\
9662     You-should-not-use-'hvlines',~'rounded-corners'~and-'corners'~at-the-same-time.\\
9663     The-output-will-not-be-reliable.
9664 }

9665 \@@_msg_new:nn { key-color-inside }
9666 {
9667     Key-deprecated.\\
9668     The-key-'color-inside'~(and-its-alias-'colortbl-like')~is-now-point-less~
9669     and-have-been-deprecated.\\
9670     You-won't-have-similar-message-till-the-end-of-the-document.
9671 }

9672 \@@_msg_new:nn { invalid-weight }
9673 {
9674     Unknown-key.\\
9675     The-key~'\l_keys_key_str'~of-your-column-X-is-unknown-and-will-be-ignored.
9676 }

9677 \@@_msg_new:nn { last-col-not-used }
9678 {
9679     Column-not-used.\\
9680     The-key~'last-col'~is-in-force-but-you-have-not-used-that-last-column~
9681     in~your~\@@_full_name_env: .~
9682     However,~you-can~go~on.
9683 }

9684 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9685 {
9686     Too-much-columns.\\
9687     In-the-row~\int_eval:n { \c@iRow },~
9688     you-try-to-use-more-columns~
9689     than-allowed-by-your~\@@_full_name_env: .
9690     \@@_message_hdotsfor: \
9691     The-maximal-number-of-columns-is~\int_eval:n { \l_@@_last_col_int - 1 }~
9692     (plus-the-exterior-columns).~This-error-is-fatal.
9693 }

9694 \@@_msg_new:nn { too-much-cols-for-matrix }
9695 {
9696     Too-much-columns.\\
9697     In-the-row~\int_eval:n { \c@iRow },~
9698     you-try-to-use-more-columns-than-allowed-by-your~\@@_full_name_env: .
9699     \@@_message_hdotsfor: \
9700     Recall-that-the-maximal-number-of-columns-for-a-matrix~
9701     (excepted-the-potential-exterior-columns)~is-fixed-by-the~
9702     LaTeX-counter~'MaxMatrixCols'.~
9703     Its-current-value-is~\int_use:N \c@MaxMatrixCols \
9704     (use~\token_to_str:N \setcounter \ to-change-that-value).~
9705     This-error-is-fatal.
9706 }

9707 \@@_msg_new:nn { too-much-cols-for-array }
9708 {
9709     Too-much-columns.\\
9710     In-the-row~\int_eval:n { \c@iRow },~

```

```

9711 ~you-try-to-use-more-columns-than-allowed-by-your~
9712 \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal-number-of-columns-is~
9713 \int_use:N \g_@@_static_num_of_col_int \
9714 \bool_if:nT
9715 { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
9716 { ~(plus-the-exterior-ones) }
9717 since-the-preamble-is~' \g_@@_user_preamble_tl '.\\
9718 This-error-is-fatal.
9719 }
9720 \@@_msg_new:nn { columns~not~used }
9721 {
9722 Columns~not~used.\\
9723 The~preamble-of~your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '~
9724 It~announces~ \int_use:N \g_@@_static_num_of_col_int \
9725 columns~but~you~only~used~ \int_use:N \c@jCol .\\
9726 The~columns~you~did~not~used~won't~be~created.\\
9727 You~won't~have~similar~warning~till~the~end~of~the~document.
9728 }
9729 \@@_msg_new:nn { empty~preamble }
9730 {
9731 Empty~preamble.\\
9732 The~preamble-of~your~ \@@_full_name_env: \ is~empty.\\
9733 This-error-is-fatal.
9734 }
9735 \@@_msg_new:nn { in~first~col }
9736 {
9737 Erroneous~use.\\
9738 You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
9739 That~command~will~be~ignored.
9740 }
9741 \@@_msg_new:nn { in~last~col }
9742 {
9743 Erroneous~use.\\
9744 You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9745 That~command~will~be~ignored.
9746 }
9747 \@@_msg_new:nn { in~first~row }
9748 {
9749 Erroneous~use.\\
9750 You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9751 That~command~will~be~ignored.
9752 }
9753 \@@_msg_new:nn { in~last~row }
9754 {
9755 Erroneous~use.\\
9756 You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9757 That~command~will~be~ignored.
9758 }
9759 \@@_msg_new:nn { TopRule~without~booktabs }
9760 {
9761 Erroneous~use.\\
9762 You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
9763 That~command~will~be~ignored.
9764 }
9765 \@@_msg_new:nn { TopRule~without~tikz }
9766 {
9767 Erroneous~use.\\
9768 You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
9769 That~command~will~be~ignored.
9770 }

```

```

9771 \@@_msg_new:nn { caption~outside~float }
9772 {
9773   Key~caption~forbidden.\\
9774   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9775   environment~(such~as~\{table\}).~This~key~will~be~ignored.
9776 }
9777 \@@_msg_new:nn { short~caption~without~caption }
9778 {
9779   You~should~not~use~the~key~'short~caption'~without~'caption'.~
9780   However,~your~'short~caption'~will~be~used~as~'caption'.
9781 }
9782 \@@_msg_new:nn { double~closing~delimiter }
9783 {
9784   Double~delimiter.\\
9785   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9786   delimiter.~This~delimiter~will~be~ignored.
9787 }
9788 \@@_msg_new:nn { delimiter~after~opening }
9789 {
9790   Double~delimiter.\\
9791   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9792   delimiter.~That~delimiter~will~be~ignored.
9793 }
9794 \@@_msg_new:nn { bad~option~for~line~style }
9795 {
9796   Bad~line~style.\\
9797   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9798   is~'standard'.~That~key~will~be~ignored.
9799 }
9800 \@@_msg_new:nn { corners~with~no~cell~nodes }
9801 {
9802   Incompatible~keys.\\
9803   You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
9804   is~in~force.\\
9805   If~you~go~on,~that~key~will~be~ignored.
9806 }
9807 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
9808 {
9809   Incompatible~keys.\\
9810   You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
9811   is~in~force.\\
9812   If~you~go~on,~those~extra~nodes~won't~be~created.
9813 }
9814 \@@_msg_new:nn { Identical~notes~in~caption }
9815 {
9816   Identical~tabular~notes.\\
9817   You~can't~put~several~notes~with~the~same~content~in~
9818   \token_to_str:N \caption \ (but~you~can~in~the~main~tabular).\\
9819   If~you~go~on,~the~output~will~probably~be~erroneous.
9820 }
9821 \@@_msg_new:nn { tabularnote~below~the~tabular }
9822 {
9823   \token_to_str:N \tabularnote \ forbidden\\
9824   You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
9825   of~your~tabular~because~the~caption~will~be~composed~below~
9826   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9827   key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .\\
9828   Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
9829   no~similar~error~will~raised~in~this~document.
9830 }

```

```

9831 \@@_msg_new:nn { Unknown~key~for~rules }
9832 {
9833   Unknown~key.\\
9834   There~is~only~two~keys~available~here:~width~and~color.\\
9835   Your~key~' \l_keys_key_str '~will~be~ignored.
9836 }
9837 \@@_msg_new:nn { Unknown~key~for~Hbrace }
9838 {
9839   Unknown~key.\\
9840   You~have~used~the~key~' \l_keys_key_str '~but~the~only~
9841   keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
9842   and~ \token_to_str:N \Vbrace \ are:~'color',~
9843   'horizontal-label(s)',~'shorten'~'shorten-end'~
9844   and~'shorten-start'.\\
9845   That~error~is~fatal.
9846 }
9847 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9848 {
9849   Unknown~key.\\
9850   There~is~only~two~keys~available~here:~
9851   'empty'~and~'not-empty'.\\
9852   Your~key~' \l_keys_key_str '~will~be~ignored.
9853 }
9854 \@@_msg_new:nn { Unknown~key~for~rotate }
9855 {
9856   Unknown~key.\\
9857   The~only~key~available~here~is~'c'.\\
9858   Your~key~' \l_keys_key_str '~will~be~ignored.
9859 }
9860 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9861 {
9862   Unknown~key.\\
9863   The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
9864   It~you~go~on,~you~will~probably~have~other~errors. \\
9865   \c_@@_available_keys_str
9866 }
9867 {
9868   The~available~keys~are~(in~alphabetic~order):~
9869   ccommand,~
9870   color,~
9871   command,~
9872   dotted,~
9873   letter,~
9874   multiplicity,~
9875   sep-color,~
9876   tikz,~and~total-width.
9877 }
9878 \@@_msg_new:nnn { Unknown~key~for~xdots }
9879 {
9880   Unknown~key.\\
9881   The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9882   \c_@@_available_keys_str
9883 }
9884 {
9885   The~available~keys~are~(in~alphabetic~order):~
9886   'color',~
9887   'horizontal(s)-labels',~
9888   'inter',~
9889   'line-style',~
9890   'radius',~
9891   'shorten',~
9892   'shorten-end'~and~'shorten-start'.

```



```

9893     }
9894 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9895 {
9896   Unknown~key.\\
9897   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9898   (and~you~try~to~use~' \l_keys_key_str ')\
9899   That~key~will~be~ignored.
9900 }
9901 \@@_msg_new:nn { label~without~caption }
9902 {
9903   You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
9904   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9905 }
9906 \@@_msg_new:nn { W~warning }
9907 {
9908   Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
9909   (row~ \int_use:N \c@iRow ).
9910 }
9911 \@@_msg_new:nn { Construct~too~large }
9912 {
9913   Construct~too~large.\\
9914   Your~command~ \token_to_str:N #1
9915   can't~be~drawn~because~your~matrix~is~too~small.\\
9916   That~command~will~be~ignored.
9917 }
9918 \@@_msg_new:nn { underscore~after~nicematrix }
9919 {
9920   Problem~with~'underscore'.\\
9921   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9922   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9923   ' \token_to_str:N \Cdots \token_to_str:N _
9924   \{ n \token_to_str:N \text \{ ~times \} \}'.
9925 }
9926 \@@_msg_new:nn { ampersand~in~light~syntax }
9927 {
9928   Ampersand~forbidden.\\
9929   You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
9930   ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9931 }
9932 \@@_msg_new:nn { double~backslash~in~light~syntax }
9933 {
9934   Double~backslash~forbidden.\\
9935   You~can't~use~ \token_to_str:N \\
9936   ~to~separate~rows~because~the~key~'light-syntax'~
9937   is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
9938   (set~by~the~key~'end-of-row').~This~error~is~fatal.
9939 }
9940 \@@_msg_new:nn { hlines~with~color }
9941 {
9942   Incompatible~keys.\\
9943   You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9944   \token_to_str:N \Block \ when~the~key~'color'~or~'draw'~is~used.\\
9945   However,~you~can~put~several~commands~ \token_to_str:N \Block.\\
9946   Your~key~will~be~discarded.
9947 }
9948 \@@_msg_new:nn { bad~value~for~baseline }
9949 {
9950   Bad~value~for~baseline.\\
9951   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
9952   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~

```

```

9953 \int_use:N \g_@@_row_total_int \ or-equal-to~'t',~'c'~or~'b'~or~of~
9954 the-form~'line-i'.\\
9955 A~value~of~1~will~be~used.
9956 }
9957 \@@_msg_new:nn { detection-of-empty-cells }
9958 {
9959 Problem-with~'not-empty'\\
9960 For-technical-reasons,~you~must~activate~
9961 'create-cell-nodes'~in~ \token_to_str:N \CodeBefore \
9962 in-order-to~use~the~key~' \l_keys_key_str '.\\
9963 That~key~will~be~ignored.
9964 }
9965 \@@_msg_new:nn { siunitx-not-loaded }
9966 {
9967 siunitx-not-loaded\\
9968 You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9969 That~error~is~fatal.
9970 }
9971 \@@_msg_new:nn { Invalid-name }
9972 {
9973 Invalid-name.\\
9974 You~can't~give~the~name~' \l_keys_value_tl '~to~a~ \token_to_str:N
9975 \SubMatrix \ of~your~ \@@_full_name_env: .\\
9976 A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9977 This~key~will~be~ignored.
9978 }
9979 \@@_msg_new:nn { Hbrace-not-allowed }
9980 {
9981 Command-not-allowed.\\
9982 You~can't~use~the~command~ \token_to_str:N #1
9983 because~you~have~not~loaded~
9984 \IfPackageLoadedTF { tikz }
9985 { the-TikZ-library~'decorations.pathreplacing'~Use~ }
9986 { TikZ~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
9987 \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\
9988 That~command~will~be~ignored.
9989 }
9990 \@@_msg_new:nn { Vbrace-not-allowed }
9991 {
9992 Command-not-allowed.\\
9993 You~can't~use~the~command~ \token_to_str:N \Vbrace \
9994 because~you~have~not~loaded~TikZ~
9995 and~the~TikZ-library~'decorations.pathreplacing'.\\
9996 Use: ~\token_to_str:N \usepackage \{tikz\}~
9997 \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
9998 That~command~will~be~ignored.
9999 }
10000 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
10001 {
10002 Wrong-line.\\
10003 You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
10004 \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
10005 number~is~not~valid.~It~will~be~ignored.
10006 }
10007 \@@_msg_new:nn { Impossible-delimiter }
10008 {
10009 Impossible-delimiter.\\
10010 It's~impossible~to~draw~the~#1~delimiter~of~your~
10011 \token_to_str:N \SubMatrix \ because~all~the~cells~are~empty~
10012 in~that~column.
10013 \bool_if:NT \l_@@_submatrix_slim_bool

```

```

10014     { ~Maybe-you-should-try-without-the-key-'slim'. } \\
10015     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
10016 }

10017 \@@_msg_new:nnn { width~without~X~columns }
10018 {
10019     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
10020     the~preamble~(' \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .\\
10021     That~key~will~be~ignored.
10022 }
10023 {
10024     This~message~is~the~message~'width~without~X~columns'~
10025     of~the~module~'nicematrix'.~
10026     The~experimented~users~can~disable~that~message~with~
10027     \token_to_str:N \msg_redirect_name:nnn .\\
10028 }
10029

10030 \@@_msg_new:nn { key~multiplicity~with~dotted }
10031 {
10032     Incompatible~keys. \\
10033     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
10034     in~a~'custom-line'.~They~are~incompatible. \\
10035     The~key~'multiplicity'~will~be~discarded.
10036 }

10037 \@@_msg_new:nn { empty~environment }
10038 {
10039     Empty~environment.\\
10040     Your~ \@@_full_name_env: \ is~empty.~This~error~is~fatal.
10041 }

10042 \@@_msg_new:nn { No~letter~and~no~command }
10043 {
10044     Erroneous~use.\\
10045     Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
10046     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10047     ~'ccommand'~(to~draw~horizontal~rules).\\
10048     However,~you~can~go~on.
10049 }

10050 \@@_msg_new:nn { Forbidden~letter }
10051 {
10052     Forbidden~letter.\\
10053     You~can't~use~the~letter~'#1'~for~a~customized~line.~
10054     It~will~be~ignored.\\
10055     The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10056 }

10057 \@@_msg_new:nn { Several~letters }
10058 {
10059     Wrong~name.\\
10060     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10061     have~used~' \l_@@_letter_str ').\\
10062     It~will~be~ignored.
10063 }

10064 \@@_msg_new:nn { Delimiter~with~small }
10065 {
10066     Delimiter~forbidden.\\
10067     You~can't~put~a~delimiter~in~the~preamble~of~your~
10068     \@@_full_name_env: \
10069     because~the~key~'small'~is~in~force.\\
10070     This~error~is~fatal.
10071 }

10072 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10073 {

```

```

10074     Unknown~cell.\\
10075     Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10076     the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10077     can't~be~executed~because~a~cell~doesn't~exist.\\
10078     This~command~ \token_to_str:N \line \ will~be~ignored.
10079 }
10080 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10081 {
10082     Duplicate~name.\\
10083     The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10084     in~this~ \@@_full_name_env: .\\
10085     This~key~will~be~ignored.\\
10086     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10087     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10088 }
10089 {
10090     The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
10091     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10092 }
10093 \@@_msg_new:nn { r~or~l~with~preamble }
10094 {
10095     Erroneous~use.\\
10096     You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
10097     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10098     your~ \@@_full_name_env: .\\
10099     This~key~will~be~ignored.
10100 }
10101 \@@_msg_new:nn { Hdotsfor~in~col~0 }
10102 {
10103     Erroneous~use.\\
10104     You~can't~use~ \token_to_str:N \Hdotsfor \ in~an~exterior~column~of~
10105     the~array.~This~error~is~fatal.
10106 }
10107 \@@_msg_new:nn { bad~corner }
10108 {
10109     Bad~corner.\\
10110     #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10111     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
10112     This~specification~of~corner~will~be~ignored.
10113 }
10114 \@@_msg_new:nn { bad~border }
10115 {
10116     Bad~border.\\
10117     \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
10118     (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
10119     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10120     also~use~the~key~'tikz'
10121     \IfPackageLoadedF { tikz }
10122     { ~if~you~load~the~LaTeX~package~'tikz' } ).\\
10123     This~specification~of~border~will~be~ignored.
10124 }
10125 \@@_msg_new:nn { TikzEveryCell~without~tikz }
10126 {
10127     TikZ~not~loaded.\\
10128     You~can't~use~ \token_to_str:N \TikzEveryCell \
10129     because~you~have~not~loaded~tikz.~
10130     This~command~will~be~ignored.
10131 }
10132 \@@_msg_new:nn { tikz~key~without~tikz }
10133 {
10134     TikZ~not~loaded.\\

```

```

10135     You-can't-use-the-key-'tikz'~for-the-command~' \token_to_str:N
10136     \Block ~because-you-have-not-loaded-tikz.~
10137     This-key-will-be-ignored.
10138   }
10139   \@@_msg_new:nn { Bad-argument-for-Block }
10140   {
10141     Bad-argument.\\
10142     The-first-mandatory-argument-of~\token_to_str:N \Block\ must~
10143     be-of-the-form~'i-j'~(or-completely-empty)~and-you-have-used:~
10144     '#1'. \\
10145     If-you-go-on,~the~\token_to_str:N \Block\ will-be-mono-cell~(as-if~
10146     the-argument-was-empty).
10147   }
10148   \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
10149   {
10150     Erroneous-use.\\
10151     In-the~ \@@_full_name_env: ,~you-must-use-the-key~
10152     'last-col'~without-value.\\
10153     However,~you-can-go-on-for-this-time~
10154     (the-value~' \l_keys_value_tl '~will-be-ignored).
10155   }
10156   \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
10157   {
10158     Erroneous-use. \\
10159     In~\token_to_str:N \NiceMatrixOptions ,~you-must-use-the-key~
10160     'last-col'~without-value. \\
10161     However,~you-can-go-on-for-this-time~
10162     (the-value~' \l_keys_value_tl '~will-be-ignored).
10163   }
10164   \@@_msg_new:nn { Block-too-large-1 }
10165   {
10166     Block-too-large. \\
10167     You-try-to-draw-a-block-in-the-cell~#1-#2-of-your-matrix-but-the-matrix-is~
10168     too-small-for-that-block. \\
10169     This-block-and-maybe-others-will-be-ignored.
10170   }
10171   \@@_msg_new:nn { Block-too-large-2 }
10172   {
10173     Block-too-large. \\
10174     The-preamble-of-your~ \@@_full_name_env: \ announces~ \int_use:N
10175     \g_@@_static_num_of_col_int \
10176     columns~but-you-use-only~ \int_use:N \c@jCol \ and-that's-why-a-block~
10177     specified-in-the-cell~#1-#2-can't-be-drawn.~You-should-add-some-ampersands~
10178     (&)~at-the-end-of-the-first-row-of-your~ \@@_full_name_env: . \\
10179     This-block-and-maybe-others-will-be-ignored.
10180   }
10181   \@@_msg_new:nn { unknown-column-type }
10182   {
10183     Bad-column-type. \\
10184     The-column-type~'#1'~in-your~ \@@_full_name_env: \
10185     is-unknown. \\
10186     This-error-is-fatal.
10187   }
10188   \@@_msg_new:nn { unknown-column-type-multicolumn }
10189   {
10190     Bad-column-type. \\
10191     The-column-type~'#1'~in-the-command~\token_to_str:N \multicolumn \
10192     ~of-your~ \@@_full_name_env: \
10193     is-unknown. \\
10194     This-error-is-fatal.
10195   }

```

```

10196 \@@_msg_new:nn { unknown-column-type-S }
10197 {
10198   Bad-column-type. \\\
10199   The-column-type-'S'~in-your- \@@_full_name_env: \ is-unknown. \\\
10200   If-you-want-to-use-the-column-type-'S'~of-siunitx,-you-should-
10201   load-that-package. \\\
10202   This-error-is-fatal.
10203 }
10204 \@@_msg_new:nn { unknown-column-type-S-multicolumn }
10205 {
10206   Bad-column-type. \\\
10207   The-column-type-'S'~in-the-command~\token_to_str:N \multicolumn \
10208   of~your~ \@@_full_name_env: \ is-unknown. \\\
10209   If-you-want-to-use-the-column-type-'S'~of-siunitx,-you-should-
10210   load-that-package. \\\
10211   This-error-is-fatal.
10212 }
10213 \@@_msg_new:nn { tabularnote~forbidden }
10214 {
10215   Forbidden-command. \\\
10216   You-can't-use-the-command~ \token_to_str:N \tabularnote \
10217   ~here.~This-command-is-available-only-in~
10218   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
10219   the-argument-of-a-command~\token_to_str:N \caption \ included-
10220   in-an-environment~\{table\}. \\\
10221   This-command-will-be-ignored.
10222 }
10223 \@@_msg_new:nn { borders~forbidden }
10224 {
10225   Forbidden-key.\\
10226   You-can't-use-the-key~'borders'~of-the-command~ \token_to_str:N \Block \
10227   because-the-option~'rounded-corners'~
10228   is~in~force~with~a~non~zero~value.\\
10229   This-key-will-be-ignored.
10230 }
10231 \@@_msg_new:nn { bottomrule~without~booktabs }
10232 {
10233   booktabs-not-loaded.\\
10234   You-can't-use-the-key~'tabular/bottomrule'~because-you-haven't~
10235   loaded~'booktabs'.\\
10236   This-key-will-be-ignored.
10237 }
10238 \@@_msg_new:nn { enumitem~not~loaded }
10239 {
10240   enumitem-not-loaded. \\\
10241   You-can't-use-the-command~ \token_to_str:N \tabularnote \
10242   ~because-you-haven't~loaded~'enumitem'. \\\
10243   All-the-commands~ \token_to_str:N \tabularnote \ will-be-
10244   ignored-in-the-document.
10245 }
10246 \@@_msg_new:nn { tikz~without~tikz }
10247 {
10248   Tikz~not-loaded. \\\
10249   You-can't-use-the-key~'tikz'~here~because-Tikz-is-not~
10250   loaded.~If-you-go-on,~that-key-will-be-ignored.
10251 }
10252 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
10253 {
10254   Tikz~not-loaded. \\\
10255   You-have-used-the-key~'tikz'~in-the-definition-of-a~
10256   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~

```

```

10257     You~can~go~on~but~you~will~have~another~error~if~you~actually~
10258     use~that~custom~line.
10259 }

10260 \@@_msg_new:nn { tikz-in~borders~without~tikz }
10261 {
10262     Tikz~not~loaded. \\
10263     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
10264     command~' \token_to_str:N \Block ')~but~tikz~is~not~loaded.~
10265     That~key~will~be~ignored.
10266 }

10267 \@@_msg_new:nn { color-in~custom~line~with~tikz }
10268 {
10269     Erroneous~use.\\
10270     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
10271     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
10272     The~key~'color'~will~be~discarded.
10273 }

10274 \@@_msg_new:nn { Wrong~last~row }
10275 {
10276     Wrong~number.\\
10277     You~have~used~'last~row= \int_use:N \l_@@_last_row_int '~but~your~
10278     \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
10279     If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
10280     last~row~but~you~should~correct~your~code.~You~can~avoid~this~
10281     problem~by~using~'last~row'~without~value~(more~compilations~
10282     might~be~necessary).
10283 }

10284 \@@_msg_new:nn { Yet~in~env }
10285 {
10286     Nested~environments.\\
10287     Environments~of~nicematrix~can't~be~nested.\\
10288     This~error~is~fatal.
10289 }

10290 \@@_msg_new:nn { Outside~math~mode }
10291 {
10292     Outside~math~mode.\\
10293     The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
10294     (and~not~in~ \token_to_str:N \vcenter ).\\
10295     This~error~is~fatal.
10296 }

10297 \@@_msg_new:nn { One~letter~allowed }
10298 {
10299     Bad~name.\\
10300     The~value~of~key~' \l_keys_key_str '~must~be~of~length~1~and~
10301     you~have~used~' \l_keys_value_tl '.\\
10302     It~will~be~ignored.
10303 }

10304 \@@_msg_new:nn { TabularNote~in~CodeAfter }
10305 {
10306     Environment~\{TabularNote\}~forbidden.\\
10307     You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
10308     but~*before*~the~ \token_to_str:N \CodeAfter . \\
10309     This~environment~\{TabularNote\}~will~be~ignored.
10310 }

10311 \@@_msg_new:nn { varwidth~not~loaded }
10312 {
10313     varwidth~not~loaded.\\
10314     You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
10315     loaded.\\
10316     Your~column~will~behave~like~'p'.

```

```

10317 }
10318 \@@_msg_new:nn { varwidth~not~loaded~in~X }
10319 {
10320   varwidth~not~loaded.\\
10321   You~can't~use~the~key~'V'~in~your~column~'X'~
10322   because~'varwidth'~is~not~loaded.\\
10323   It~will~be~ignored. \\
10324 }
10325 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
10326 {
10327   Unknown~key.\\
10328   Your~key~' \l_keys_key_str '~is~unknown~for~a~rule.\\
10329   \c_@@_available_keys_str
10330 }
10331 {
10332   The~available~keys~are~(in~alphabetic~order):~
10333   color,~
10334   dotted,~
10335   multiplicity,~
10336   sep-color,~
10337   tikz,~and~total~width.
10338 }
10339
10340 \@@_msg_new:nnn { Unknown~key~for~Block }
10341 {
10342   Unknown~key. \\
10343   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10344   \token_to_str:N \Block . \\
10345   It~will~be~ignored. \\
10346   \c_@@_available_keys_str
10347 }
10348 {
10349   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10350   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line~width,~name,~
10351   opacity,~rounded~corners,~r,~respect~arraystretch,~t,~T,~tikz,~transparent~
10352   and~vlines.
10353 }
10354 \@@_msg_new:nnn { Unknown~key~for~Brace }
10355 {
10356   Unknown~key.\\
10357   The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
10358   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
10359   It~will~be~ignored. \\
10360   \c_@@_available_keys_str
10361 }
10362 {
10363   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
10364   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
10365   right~shorten)~and~yshift.
10366 }
10367 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10368 {
10369   Unknown~key.\\
10370   The~key~' \l_keys_key_str '~is~unknown.\\
10371   It~will~be~ignored. \\
10372   \c_@@_available_keys_str
10373 }
10374 {
10375   The~available~keys~are~(in~alphabetic~order):~
10376   delimiters/color,~
10377   rules~(with~the~subkeys~'color'~and~'width'),~
10378   sub-matrix~(several~subkeys)~

```



```

10379     and~xdots~(several~subkeys).~
10380     The~latter~is~for~the~command~ \token_to_str:N \line .
10381 }

10382 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10383 {
10384     Unknown~key.\\
10385     The~key~' \l_keys_key_str '~is~unknown.\\
10386     It~will~be~ignored. \\
10387     \c_@@_available_keys_str
10388 }
10389 {
10390     The~available~keys~are~(in~alphabetic~order):~
10391     create~cell~nodes,~
10392     delimiters/color~and~
10393     sub~matrix~(several~subkeys).
10394 }

10395 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10396 {
10397     Unknown~key.\\
10398     The~key~' \l_keys_key_str '~is~unknown.\\
10399     That~key~will~be~ignored. \\
10400     \c_@@_available_keys_str
10401 }
10402 {
10403     The~available~keys~are~(in~alphabetic~order):~
10404     'delimiters/color',~
10405     'extra-height',~
10406     'hlines',~
10407     'hvlines',~
10408     'left-xshift',~
10409     'name',~
10410     'right-xshift',~
10411     'rules'~(with~the~subkeys~'color'~and~'width'),~
10412     'slim',~
10413     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10414     and~'right-xshift').\\
10415 }

10416 \@@_msg_new:nnn { Unknown~key~for~notes }
10417 {
10418     Unknown~key.\\
10419     The~key~' \l_keys_key_str '~is~unknown.\\
10420     That~key~will~be~ignored. \\
10421     \c_@@_available_keys_str
10422 }
10423 {
10424     The~available~keys~are~(in~alphabetic~order):~
10425     bottomrule,~
10426     code~after,~
10427     code~before,~
10428     detect~duplicates,~
10429     enumitem~keys,~
10430     enumitem~keys~para,~
10431     para,~
10432     label~in~list,~
10433     label~in~tabular~and~
10434     style.
10435 }

10436 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10437 {
10438     Unknown~key.\\
10439     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10440     \token_to_str:N \RowStyle . \\

```

```

10441     That~key~will~be~ignored. \\
10442     \c_@@_available_keys_str
10443 }
10444 {
10445     The~available~keys~are~(in~alphabetic~order):~
10446     bold,~
10447     cell-space-top-limit,~
10448     cell-space-bottom-limit,~
10449     cell-space-limits,~
10450     color,~
10451     fill~(alias:~rowcolor),~
10452     nb-rows,~
10453     opacity~and~
10454     rounded-corners.
10455 }
10456 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10457 {
10458     Unknown~key.\\
10459     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10460     \token_to_str:N \NiceMatrixOptions . \\
10461     That~key~will~be~ignored. \\
10462     \c_@@_available_keys_str
10463 }
10464 {
10465     The~available~keys~are~(in~alphabetic~order):~
10466     &-in-blocks,~
10467     allow-duplicate-names,~
10468     ampersand-in-blocks,~
10469     caption-above,~
10470     cell-space-bottom-limit,~
10471     cell-space-limits,~
10472     cell-space-top-limit,~
10473     code-for-first-col,~
10474     code-for-first-row,~
10475     code-for-last-col,~
10476     code-for-last-row,~
10477     corners,~
10478     custom-key,~
10479     create-extra-nodes,~
10480     create-medium-nodes,~
10481     create-large-nodes,~
10482     custom-line,~
10483     delimiters~(several~subkeys),~
10484     end-of-row,~
10485     first-col,~
10486     first-row,~
10487     hlines,~
10488     hvlines,~
10489     hvlines-except-borders,~
10490     last-col,~
10491     last-row,~
10492     left-margin,~
10493     light-syntax,~
10494     light-syntax-expanded,~
10495     matrix/columns-type,~
10496     no-cell-nodes,~
10497     notes~(several~subkeys),~
10498     nullify-dots,~
10499     pgf-node-code,~
10500     renew-dots,~
10501     renew-matrix,~
10502     respect-arraystretch,~
10503     rounded-corners,~

```

```

10504     right-margin,~
10505     rules~(with~the~subkeys~'color'~and~'width'),~
10506     small,~
10507     sub-matrix~(several~subkeys),~
10508     vl_lines,~
10509     xdots~(several~subkeys).
10510 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

10511 \@@_msg_new:nnn { Unknown-key-for~NiceArray }
10512 {
10513   Unknown-key.\\
10514   The~key~' \l_keys_key_str '~is-unknown-for~the~environment~
10515   \{NiceArray\}. \\
10516   That~key~will~be~ignored. \\
10517   \c_@@_available_keys_str
10518 }
10519 {
10520   The~available~keys~are~(in~alphabetic~order):~
10521   &-in-blocks,~
10522   ampersand-in-blocks,~
10523   b,~
10524   baseline,~
10525   c,~
10526   cell-space-bottom-limit,~
10527   cell-space-limits,~
10528   cell-space-top-limit,~
10529   code-after,~
10530   code-for-first-col,~
10531   code-for-first-row,~
10532   code-for-last-col,~
10533   code-for-last-row,~
10534   columns-width,~
10535   corners,~
10536   create-extra-nodes,~
10537   create-medium-nodes,~
10538   create-large-nodes,~
10539   extra-left-margin,~
10540   extra-right-margin,~
10541   first-col,~
10542   first-row,~
10543   hlines,~
10544   hvlines,~
10545   hvlines-except-borders,~
10546   last-col,~
10547   last-row,~
10548   left-margin,~
10549   light-syntax,~
10550   light-syntax-expanded,~
10551   name,~
10552   no-cell-nodes,~
10553   nullify-dots,~
10554   pgf-node-code,~
10555   renew-dots,~
10556   respect-arraystretch,~
10557   right-margin,~
10558   rounded-corners,~
10559   rules~(with~the~subkeys~'color'~and~'width'),~
10560   small,~
10561   t,~
10562   vl_lines,~
10563   xdots/color,~
10564   xdots/shorten-start,~

```

```

10565     xdots/shorten-end,~
10566     xdots/shorten-and~
10567     xdots/line-style.
10568 }

```

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`).

```

10569 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10570 {
10571   Unknown~key.\\
10572   The~key~' \l_keys_key_str '~is~unknown~for~the~
10573   \@@_full_name_env: . \\
10574   That~key~will~be~ignored. \\
10575   \c_@@_available_keys_str
10576 }
10577 {
10578   The~available~keys~are~(in~alphabetic~order):~
10579   &~in~blocks,~
10580   ampersand~in~blocks,~
10581   b,~
10582   baseline,~
10583   c,~
10584   cell-space-bottom-limit,~
10585   cell-space-limits,~
10586   cell-space-top-limit,~
10587   code-after,~
10588   code-for-first-col,~
10589   code-for-first-row,~
10590   code-for-last-col,~
10591   code-for-last-row,~
10592   columns-type,~
10593   columns-width,~
10594   corners,~
10595   create-extra-nodes,~
10596   create-medium-nodes,~
10597   create-large-nodes,~
10598   extra-left-margin,~
10599   extra-right-margin,~
10600   first-col,~
10601   first-row,~
10602   hlines,~
10603   hvlines,~
10604   hvlines-except-borders,~
10605   l,~
10606   last-col,~
10607   last-row,~
10608   left-margin,~
10609   light-syntax,~
10610   light-syntax-expanded,~
10611   name,~
10612   no-cell-nodes,~
10613   nullify-dots,~
10614   pgf-node-code,~
10615   r,~
10616   renew-dots,~
10617   respect-arraystretch,~
10618   right-margin,~
10619   rounded-corners,~
10620   rules~(with~the~subkeys~'color'~and~'width'),~
10621   small,~
10622   t,~
10623   vlimes,~
10624   xdots/color,~
10625   xdots/shorten-start,~

```

```

10626     xdots/shorten-end,~
10627     xdots/shorten-and~
10628     xdots/line-style.
10629 }
10630 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10631 {
10632     Unknown~key.\\
10633     The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
10634     \{NiceTabular\}. \\
10635     That~key~will~be~ignored. \\
10636     \c_@@_available_keys_str
10637 }
10638 {
10639     The~available~keys~are~(in~alphabetic~order):~
10640     &~in~blocks,~
10641     ampersand~in~blocks,~
10642     b,~
10643     baseline,~
10644     c,~
10645     caption,~
10646     cell-space-bottom-limit,~
10647     cell-space-limits,~
10648     cell-space-top-limit,~
10649     code-after,~
10650     code-for-first-col,~
10651     code-for-first-row,~
10652     code-for-last-col,~
10653     code-for-last-row,~
10654     columns-width,~
10655     corners,~
10656     custom-line,~
10657     create-extra-nodes,~
10658     create-medium-nodes,~
10659     create-large-nodes,~
10660     extra-left-margin,~
10661     extra-right-margin,~
10662     first-col,~
10663     first-row,~
10664     hlines,~
10665     hvlines,~
10666     hvlines-except-borders,~
10667     label,~
10668     last-col,~
10669     last-row,~
10670     left-margin,~
10671     light-syntax,~
10672     light-syntax-expanded,~
10673     name,~
10674     no-cell-nodes,~
10675     notes~(several~subkeys),~
10676     nullify-dots,~
10677     pgf-node-code,~
10678     renew-dots,~
10679     respect-arraystretch,~
10680     right-margin,~
10681     rounded-corners,~
10682     rules~(with~the~subkeys~'color'~and~'width'),~
10683     short-caption,~
10684     t,~
10685     tabularnote,~
10686     vlides,~
10687     xdots/color,~
10688     xdots/shorten-start,~

```

```

10689     xdots/shorten-end,~
10690     xdots/shorten-and~
10691     xdots/line-style.
10692 }

10693 \@@_msg_new:nnn { Duplicate-name }
10694 {
10695     Duplicate-name.\\
10696     The-name~' \l_keys_value_tl '~is-already-used-and-you-shouldn't-use~
10697     the-same-environment-name-twice.~You-can-go-on,~but,~
10698     maybe,~you-will-have-incorrect-results-especially~
10699     if-you-use~'columns-width=auto'.~If-you-don't-want-to-see-this~
10700     message-again,~use-the-key~'allow-duplicate-names'~in~
10701     ' \token_to_str:N \NiceMatrixOptions '~\\
10702     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10703     { For-a-list-of-the-names-already-used,~type-H~<return>. }
10704 }
10705 {
10706     The-names-already-defined-in-this-document-are:~
10707     \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
10708 }

10709 \@@_msg_new:nn { Option-auto-for-columns-width }
10710 {
10711     Erroneous-use.\\
10712     You-can't-give-the-value~'auto'~to-the-key~'columns-width'~here.~
10713     That-key-will-be-ignored.
10714 }

10715 \@@_msg_new:nn { NiceTabularX-without-X }
10716 {
10717     NiceTabularX-without-X.\\
10718     You-should-not-use~\{NiceTabularX\}~without-X-columns.\\
10719     However,~you-can-go-on.
10720 }

10721 \@@_msg_new:nn { Preamble-forgotten }
10722 {
10723     Preamble-forgotten.\\
10724     You-have-probably-forgotten-the-preamble-of-your~
10725     \@@_full_name_env: . \\
10726     This-error-is-fatal.
10727 }

10728 \@@_msg_new:nn { Invalid-col-number }
10729 {
10730     Invalid-column-number.\\
10731     A-color-instruction-in-the~ \token_to_str:N \CodeBefore \
10732     specifies-a-column-which-is-outside-the-array.~It-will-be-ignored.
10733 }

10734 \@@_msg_new:nn { Invalid-row-number }
10735 {
10736     Invalid-row-number.\\
10737     A-color-instruction-in-the~ \token_to_str:N \CodeBefore \
10738     specifies-a-row-which-is-outside-the-array.~It-will-be-ignored.
10739 }

10740 \@@_define_com:NNN p ( )
10741 \@@_define_com:NNN b [ ]
10742 \@@_define_com:NNN v | |
10743 \@@_define_com:NNN V \l \l
10744 \@@_define_com:NNN B \{ \}

```

Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	3
4	Parameters	9
5	The command <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>	51
10	The environment <code>{NiceArrayWithDelims}</code>	55
11	Construction of the preamble of the array	60
12	The redefinition of <code>\multicolumn</code>	76
13	The environment <code>{NiceMatrix}</code> and its variants	94
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	95
15	After the construction of the array	96
16	We draw the dotted lines	103
17	The actual instructions for drawing the dotted lines with Tikz	118
18	User commands available in the new environments	123
19	The command <code>\line</code> accessible in code-after	129
20	The command <code>\RowStyle</code>	131
21	Colors of cells, rows and columns	134
22	The vertical and horizontal rules	146
23	The empty corners	162
24	The environment <code>{NiceMatrixBlock}</code>	164
25	The extra nodes	166
26	The blocks	170
27	How to draw the dotted lines transparently	195
28	Automatic arrays	195
29	The redefinition of the command <code>\dotfill</code>	197
30	The command <code>\diagbox</code>	197

31	The keyword <code>\CodeAfter</code>	198
32	The delimiters in the preamble	199
33	The command <code>\SubMatrix</code>	200
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	209
35	The commands <code>HBrace</code> et <code>VBrace</code>	212
36	The command <code>TikzEveryCell</code>	215
37	The command <code>\ShowCellNames</code>	216
38	We process the options at package loading	218
39	About the package underscore	220
40	Error messages of the package	220