

# The code of the package `nicematrix`\*

F. Pantigny  
fpantigny@wanadoo.fr

October 29, 2024

## 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 traduction: `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 \RequirePackage{l3keys2e}
4 \ProvidesExplPackage
5   {nicematrix}
6   {\myfiledate}
7   {\myfileversion}
8   {Enhanced arrays with the help of PGF/TikZ}
9 \ProvideDocumentCommand{\IfPackageLoadedT}{mm}
10  {\IfPackageLoadedTF{#1}{#2}{}}
11
12 \ProvideDocumentCommand{\IfPackageLoadedF}{mm}
13  {\IfPackageLoadedTF{#1}{}{#2}}
```

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

```
14 \RequirePackage { amsmath }

15 \RequirePackage { array }
```

---

\*This document corresponds to the version 6.29a of `nicematrix`, at the date of 2024/10/29.

In the version 2.6a of array, important modifications have been done for the Tagging Project.

```

16 \bool_const:Nn \c_@@_tagging_array_bool
17 { \IfPackageAtLeastTF { array } { 2024/05/01 } \c_true_bool \c_false_bool }
18 \bool_const:Nn \c_@@_testphase_table_bool
19 { \IfPackageLoadedTF { latex-lab-testphase-table } \c_true_bool \c_false_bool
20 }

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

```

With Overleaf, 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).

```

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

```

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

```

35 \cs_new_protected:Npn \@@_error_or_warning:n
36 { \bool_if:NTF \g_@@_messages_for_Overleaf_bool \@@_warning:n \@@_error:n }

```

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".

```

37 \bool_new:N \g_@@_messages_for_Overleaf_bool
38 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
39 {
40   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
41   || \str_if_eq_p:ee \c_sys_jobname_str { output } % for Overleaf
42 }

```

```

43 \cs_new_protected:Npn \@@_msg_redirect_name:nn
44 { \msg_redirect_name:nnn { nicematrix } }
45 \cs_new_protected:Npn \@@_gredirect_none:n #1
46 {
47   \group_begin:
48   \globaldefs = 1
49   \@@_msg_redirect_name:nn { #1 } { none }
50   \group_end:
51 }
52 \cs_new_protected:Npn \@@_err_gredirect_none:n #1
53 {
54   \@@_error:n { #1 }
55   \@@_gredirect_none:n { #1 }
56 }
57 \cs_new_protected:Npn \@@_warning_gredirect_none:n #1
58 {
59   \@@_warning:n { #1 }
60   \@@_gredirect_none:n { #1 }
61 }

```

We will delete in the future the following lines which are only a security.

```
62 \cs_set:Npn \int_if_zero:NT #1 { \int_compare:nNnT #1 = \c_zero_int }
63 \cs_set:Npn \int_if_zero:NTF #1 { \int_compare:nNnTF #1 = \c_zero_int }

64 \@@_msg_new:nn { mdwtab-loaded }
65 {
66   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
67   This~error~is~fatal.
68 }

69 \hook_gput_code:nnn { begindocument / end } { . }
70 { \IfPackageLoadedT { mdwtab } { \@@_fatal:n { mdwtab-loaded } } }
```

## 2 Collecting options

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

*Exemple :*

```
\@@_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`).

```
71 \cs_new_protected:Npn \@@_collect_options:n #1
72 {
73   \peek_meaning:NTF [
74     { \@@_collect_options:nw { #1 } }
75     { #1 { } }
76 }
```

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

```
77 \NewDocumentCommand \@@_collect_options:nw { m r[] }
78 { \@@_collect_options:nn { #1 } { #2 } }
79
80 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
81 {
82   \peek_meaning:NTF [
83     { \@@_collect_options:nw { #1 } { #2 } }
84     { #1 { #2 } }
85 }
86
87 \cs_new_protected:Npn \@@_collect_options:nw #1#2[#3]
88 { \@@_collect_options:nn { #1 } { #2 , #3 } }
```

### 3 Technical definitions

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

```

89 \tl_const:Nn \c_@@_b_tl { b }
90 \tl_const:Nn \c_@@_c_tl { c }
91 \tl_const:Nn \c_@@_l_tl { l }
92 \tl_const:Nn \c_@@_r_tl { r }
93 \tl_const:Nn \c_@@_all_tl { all }
94 \tl_const:Nn \c_@@_dot_tl { . }
95 \tl_const:Nn \c_@@_default_tl { default }
96 \tl_const:Nn \c_@@_star_tl { * }
97 \str_const:Nn \c_@@_star_str { * }
98 \str_const:Nn \c_@@_r_str { r }
99 \str_const:Nn \c_@@_c_str { c }
100 \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).

```

101 \tl_new:N \l_@@_argspec_tl
102 \cs_generate_variant:Nn \seq_set_split:Nnn { N o }
103 \cs_generate_variant:Nn \str_lowercase:n { o }
104 \cs_generate_variant:Nn \str_set:Nn { N o }
105 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
106 \prg_generate_conditional_variant:Nnn \clist_if_in:Nn { N e } { T , F, TF }
107 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { e } { T }
108 \prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
109 \cs_generate_variant:Nn \dim_min:nn { v }
110 \cs_generate_variant:Nn \dim_max:nn { v }

111 \hook_gput_code:nnn { begindocument } { . }
112 {
113   \IfPackageLoadedTF { tikz }
114   {

```

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.

```

115   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
116   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
117 }
118 {
119   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
120   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
121 }
122 }

```

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 2024, the current version `revtex4-2` is 4.2f (compatible with `booktabs`).

```

123 \IfClassLoadedTF { revtex4-1 }
124 { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
125 {
126   \IfClassLoadedTF { revtex4-2 }
127   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
128   {

```

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

```

129     \cs_if_exist:NT \rvtx@ifformat@geq
130     { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
131     { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
132   }
133 }

```

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

```

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

```

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

```

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

```

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

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

```

159 \hook_gput_code:nnn { begindocument } { . }
160 {
161   \IfPackageLoadedT { booktabs }
162   { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
163 }
164 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
165 {
166   \cs_set_eq:NN \@@_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`).

```

167   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
168   {

```

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

```
169     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
170     { \@@_old_pgfulil@check@rerun { ##1 } { ##2 } }
171   }
172 }
```

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

```
173 \hook_gput_code:nnn { begindocument } { . }
174 {
175   \IfPackageLoadedF { colortbl }
176   {
```

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.

```
177     \cs_set_protected:Npn \CT@arc@ { }
178     \cs_set_nopar:Npn \arrayrulecolor #1 # { \CT@arc { #1 } }
179     \cs_set_nopar:Npn \CT@arc #1 #2
180     {
181       \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
182       { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
183     }
```

Idem for `\CT@drs@`.

```
184     \cs_set_nopar:Npn \doublerulesepcolor #1 # { \CT@drs { #1 } }
185     \cs_set_nopar:Npn \CT@drs #1 #2
186     {
187       \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
188       { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
189     }
190     \cs_set_nopar:Npn \hline
191     {
192       \noalign { \ifnum 0 = ` } \fi
193       \cs_set_eq:NN \hskip \vskip
194       \cs_set_eq:NN \vrule \hrule
195       \cs_set_eq:NN \@width \@height
196       { \CT@arc@ \vline }
197       \futurelet \reserved@a
198       \@xhline
199     }
200   }
201 }
```

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.

```
202 \cs_set_nopar:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
203 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
204 {
205   \int_if_zero:nT \l_@@_first_col_int { \omit & }
206   \int_compare:nNnT { #1 } > \c_one_int
207   { \multispan { \int_eval:n { #1 - 1 } } & }
208   \multispan { \int_eval:n { #2 - #1 + 1 } }
209   {
210     \CT@arc@
211     \leaders \hrule \@height \arrayrulewidth \hfill
```

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

```
212     \skip_horizontal:N \c_zero_dim
213   }
```

---

<sup>1</sup>See question 99041 on TeX StackExchange.

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`.

```

214   \everycr { }
215   \cr
216   \noalign { \skip_vertical:N -\arrayrulewidth }
217 }

```

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.

```

218 \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`.

```

219 { \@@_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*.

```

220 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
221 \cs_generate_variant:Nn \@@_cline_i:nn { e }
222 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
223 {
224   \tl_if_empty:nTF { #3 }
225     { \@@_cline_iii:w #1|#2-#2 \q_stop }
226     { \@@_cline_ii:w #1|#2-#3 \q_stop }
227 }
228 \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
229 { \@@_cline_iii:w #1|#2-#3 \q_stop }
230 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
231 {

```

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).

```

232   \int_compare:nNnT { #1 } < { #2 }
233     { \multispan { \int_eval:n { #2 - #1 } } & }
234   \multispan { \int_eval:n { #3 - #2 + 1 } }
235   {
236     \CT@arc@
237     \leaders \hrule \@height \arrayrulewidth \hfill
238     \skip_horizontal:N \c_zero_dim
239   }

```

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

```

240   \peek_meaning_remove_ignore_spaces:NNTF \cline
241     { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
242     { \everycr { } \cr }
243 }

```

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

```

244 \cs_set_eq:NN \@@_math_toggle: \c_math_toggle_token

245 \cs_generate_variant:Nn \@@_set_CT@arc@:n { o }
246 \cs_new_protected:Npn \@@_set_CT@arc@:n #1
247 {
248   \tl_if_blank:nF { #1 }
249   {
250     \tl_if_head_eq_meaning:nNTF { #1 } [
251       { \cs_set_nopar:Npn \CT@arc@ { \color #1 } }
252       { \cs_set_nopar:Npn \CT@arc@ { \color { #1 } } }
253     ]
254 }

```

```

255 \cs_generate_variant:Nn \@@_set_CT@drsc@:n { o }
256 \cs_new_protected:Npn \@@_set_CT@drsc@:n #1
257 {
258   \tl_if_head_eq_meaning:nNTF { #1 } [
259     { \cs_set_nopar:Npn \CT@drsc@ { \color #1 } }
260     { \cs_set_nopar:Npn \CT@drsc@ { \color { #1 } } }
261   ]

```

The following command must *not* be protected since it will be used to write instructions in the (internal) `\CodeBefore`.

```

262 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }
263 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
264 {
265   \tl_if_head_eq_meaning:nNTF { #2 } [
266     { #1 #2 }
267     { #1 { #2 } }
268   ]

```

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

```

269 \cs_generate_variant:Nn \@@_color:n { o }
270 \cs_new_protected:Npn \@@_color:n #1
271 { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }

```

```

272 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
273 {
274   \tl_set_rescan:Nno
275     #1
276     {
277       \char_set_catcode_other:N >
278       \char_set_catcode_other:N <
279     }
280   #1
281 }

```

## 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.

```

282 \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).

```

283 \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.

```

284 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
285 { \int_use:N \g_@@_env_int }

```

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

```

286 \cs_new_protected:Npn \@@_qpoint:n #1
287 { \pgfpointanchor { \@@_env: - #1 } { center } }

```



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

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

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

```
289 \bool_new:N \g_@@_delims_bool
290 \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.

```
291 \bool_new:N \l_@@_preamble_bool
292 \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.

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

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

```
294 \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.

```
295 \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).

```
296 \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.).

```
297 \dim_new:N \l_@@_col_width_dim
298 \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.

```
299 \int_new:N \g_@@_row_total_int
300 \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).

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

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

```
302 \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.

```
303 \tl_new:N \l_@@_hpos_cell_tl
304 \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`.

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

Idem for the mono-row blocks.

```
306 \dim_new:N \g_@@_blocks_ht_dim
307 \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}`).

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

The sequence `\g_@@_names_seq` 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`.

```
309 \seq_new:N \g_@@_names_seq
```

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.

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

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

```
311 \bool_new:N \l_@@_notes_detect_duplicates_bool
312 \bool_set_true:N \l_@@_notes_detect_duplicates_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.

```
313 \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).

```
314 \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).

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

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

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

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

```
317 \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.

```
318 \bool_new:N \l_@@_X_bool
319 \bool_new:N \g_@@_caption_finished_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 contain of that token list will be written on the `aux` file at the end of the environment (in an instruction `\tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }`).

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

During the second run, if informations concerning the current environment has been found in the aux file, the following flag will be raised.

```
321 \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 informations about the size of the array.

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

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

```
324 \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}`).

```
325 \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`).

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

For `\multicolumn`.

```
327 \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`.

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

```
329 \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 `:`.

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

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

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

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

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

```
334 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
335 {
```

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

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

```
338   \fi:
```

```
339 }
```

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.

```
340 \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”.

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

```
342 \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`).

```
343 \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*.

```

344 \tl_new:N \g_@@_com_or_env_str
345 \tl_gset:Nn \g_@@_com_or_env_str { environment }

346 \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)`.

```

347 \cs_new:Npn \@@_full_name_env:
348 {
349   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
350   { command \space \c_backslash_str \g_@@_name_env_str }
351   { environment \space \{ \g_@@_name_env_str \} }
352 }

```

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

```

353 \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.

```

354 \tl_new:N \l_@@_pgf_node_code_tl

```

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

```

355 \tl_new:N \g_@@_pre_code_before_tl
356 \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 splitted in two parts because we want to control the order of execution of some instructions.

```

357 \tl_new:N \g_@@_pre_code_after_tl
358 \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.

```

359 \bool_new:N \l_@@_in_code_after_bool

```

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

```

360 \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.

```

361 \int_new:N \l_@@_old_iRow_int
362 \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).

```

363 \seq_new:N \l_@@_custom_line_commands_seq

```

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

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

The sum of the weights of all the X-columns in the preamble. The weight of a X-column is given as an optional argument between square brackets. The default value, of course, is 1.

```
365 \int_new:N \g_@@_total_X_weight_int
```

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 (the width of a column of weight  $n$  will be that dimension multiplied by  $n$ ). That value is computed after the construction of the array during the first compilation in order to be used in the following run.

```
366 \bool_new:N \l_@@_X_columns_aux_bool
367 \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.

```
368 \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).

```
369 \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).

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

The use of `\l_@@_code_before_tl` is not clear. Maybe that with the evolutions of `nicematrix`, it has become obsolete. We should have a look at that.

```
371 \tl_new:N \l_@@_code_before_tl
372 \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).

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

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

```
374 \dim_new:N \l_@@_x_initial_dim
375 \dim_new:N \l_@@_y_initial_dim
376 \dim_new:N \l_@@_x_final_dim
377 \dim_new:N \l_@@_y_final_dim
```

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

```
378 \dim_new:N \l_@@_tmpc_dim
379 \dim_new:N \l_@@_tmpd_dim
380 \dim_new:N \l_@@_tmpe_dim
381 \dim_new:N \l_@@_tmpf_dim

382 \dim_new:N \g_@@_dp_row_zero_dim
383 \dim_new:N \g_@@_ht_row_zero_dim
384 \dim_new:N \g_@@_ht_row_one_dim
385 \dim_new:N \g_@@_dp_ante_last_row_dim
386 \dim_new:N \g_@@_ht_last_row_dim
387 \dim_new:N \g_@@_dp_last_row_dim
```

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

```
388 \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”.

```
389 \dim_new:N \g_@@_width_last_col_dim
390 \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.

```
391 \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!).

```
392 \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).

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}`.

```
393 \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.

```
394 \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`).

```
395 \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`).

```
396 \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.

```
397 \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).

```
398 \seq_new:N \g_@@_multicolumn_cells_seq
399 \seq_new:N \g_@@_multicolumn_sizes_seq
```

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).

```
400 \int_new:N \l_@@_row_min_int
401 \int_new:N \l_@@_row_max_int
402 \int_new:N \l_@@_col_min_int
403 \int_new:N \l_@@_col_max_int
```

The following counters will be used when drawing the rules.

```
404 \int_new:N \l_@@_start_int
405 \int_set_eq:NN \l_@@_start_int \c_one_int
406 \int_new:N \l_@@_end_int
407 \int_new:N \l_@@_local_start_int
408 \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.

```
409 \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).

```
410 \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`.

```
411 \tl_new:N \l_@@_fill_tl
412 \tl_new:N \l_@@_opacity_tl
413 \tl_new:N \l_@@_draw_tl
414 \seq_new:N \l_@@_tikz_seq
415 \clist_new:N \l_@@_borders_clist
416 \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.

```
417 \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`.

```
418 \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.

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

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

```
420 \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`).

```
421 \str_new:N \l_@@_hpos_block_str
422 \str_set:Nn \l_@@_hpos_block_str { c }
423 \bool_new:N \l_@@_hpos_of_block_cap_bool
424 \bool_new:N \l_@@_p_block_bool
```

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

```
425 \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).

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

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

```
427 \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).

```
428 \bool_new:N \l_@@_vlines_block_bool
```

```
429 \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.

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

```
431 \dim_new:N \l_@@_submatrix_extra_height_dim
```

```
432 \dim_new:N \l_@@_submatrix_left_xshift_dim
```

```
433 \dim_new:N \l_@@_submatrix_right_xshift_dim
```

```
434 \clist_new:N \l_@@_hlines_clist
```

```
435 \clist_new:N \l_@@_vlines_clist
```

```
436 \clist_new:N \l_@@_submatrix_hlines_clist
```

```
437 \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}`).

```
438 \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.

```
439 \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`).

```
440 \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.

```
441 \int_new:N \l_@@_first_row_int
442 \int_set:Nn \l_@@_first_row_int 1
```

- **First column**

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

```
443 \int_new:N \l_@@_first_col_int
444 \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).

```
445 \int_new:N \l_@@_last_row_int
446 \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”.<sup>2</sup>

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

Idem for `\l_@@_last_col_without_value_bool`

```
448 \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`.

```
449 \int_new:N \l_@@_last_col_int
450 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

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

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

```
451 \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`).

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

## Some utilities

```
453 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
454 {
```

---

<sup>2</sup>We 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.

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

```

455   \cs_set_nopar:Npn \l_tmpa_tl { #1 }
456   \cs_set_nopar:Npn \l_tmpb_tl { #2 }
457 }

```

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.

```

458 \cs_new_protected:Npn \@_expand_clist:N #1
459 {
460   \clist_if_in:NnF #1 { all }
461   {
462     \clist_clear:N \l_tmpa_clist
463     \clist_map_inline:Nn #1
464     {

```

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

```

465     \tl_if_in:nnTF { ##1 } { - }
466     { \@_cut_on_hyphen:w ##1 \q_stop }
467     {

```

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

```

468     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
469     \cs_set_nopar:Npn \l_tmpb_tl { ##1 }
470   }
471   \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
472   { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
473 }
474 \tl_set_eq:NN #1 \l_tmpa_clist
475 }
476 }

```

The following internal parameters are for:

- `\Ldots` with both extremities open (and hence also `\Hdotsfor` in an exterior row;
- `\Vdots` with both extremities open (and hence also `\Vdotsfor` in an exterior column;
- 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.

```

477 \hook_gput_code:nnn { begindocument } { . }
478 {
479   \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
480   \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
481   \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
482 }

```

## 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`.<sup>3</sup>
  - 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 `\c_novalue_tl`).
  - 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.

```
483 \newcounter { tabularnote }
```

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

```
484 \int_new:N \g_@@_tabularnote_int
485 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }
486 \seq_new:N \g_@@_notes_seq
487 \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.

```
488 \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.

```
489 \seq_new:N \l_@@_notes_labels_seq
490 \newcounter{nicematrix_draft}
491 \cs_new_protected:Npn \@@_notes_format:n #1
492 {
493   \setcounter { nicematrix_draft } { #1 }
494   \@@_notes_style:n { nicematrix_draft }
495 }
```

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

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

---

<sup>3</sup>More precisely, it's the number of tabular notes which do not use the optional argument of `\tabularnote`.

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

```
497 \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`.

```
498 \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`.

```
499 \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`).

```
500 \hook_gput_code:nnn { begindocument } { . }
501 {
502   \IfPackageLoadedTF { enumitem }
503   {
```

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.

```
504     \newlist { tabularnotes } { enumerate } { 1 }
505     \setlist [ tabularnotes ]
506     {
507       topsep = 0pt ,
508       noitemsep ,
509       leftmargin = * ,
510       align = left ,
511       labelsep = 0pt ,
512       label =
513         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
514     }
515     \newlist { tabularnotes* } { enumerate* } { 1 }
516     \setlist [ tabularnotes* ]
517     {
518       afterlabel = \nobreak ,
519       itemjoin = \quad ,
520       label =
521         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
522     }
```

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`.

```
523   \NewDocumentCommand \tabularnote { o m }
524   {
525     \bool_lazy_or:nnT { \cs_if_exist_p:N \@capytype } \l_@@_in_env_bool
526     {
527       \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } \l_@@_in_env_bool
528       { \@@_error:n { tabularnote-forbidden } }
529       {
530         \bool_if:NTF \l_@@_in_caption_bool
531         \@@_tabularnote_caption:nn
532         \@@_tabularnote:nn
533         { #1 } { #2 }
534       }
535     }
```

```

536     }
537   }
538   {
539     \NewDocumentCommand \tabularnote { o m }
540     {
541       \@@_error_or_warning:n { enumitem-not-loaded }
542       \@@_gredirect_none:n { enumitem-not-loaded }
543     }
544   }
545 }

546 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
547 { \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 `\c_novalue_tl`) and #2 is the mandatory argument of `\tabularnote`.

```

548 \cs_new_protected:Npn \@@_tabularnote:mn #1 #2
549 {

```

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).

```

550   \int_zero:N \l_tmpa_int
551   \bool_if:NT \l_@@_notes_detect_duplicates_bool
552   {

```

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 `\c_novalue_tl`.

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`.

```

553   \int_zero:N \l_tmpb_int
554   \seq_map_indexed_inline:Nn \g_@@_notes_seq
555   {
556     \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
557     \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
558     {
559       \tl_if_novalue:nTF { #1 }
560       { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
561       { \int_set:Nn \l_tmpa_int { ##1 } }
562       \seq_map_break:
563     }
564   }
565   \int_if_zero:nF \l_tmpa_int
566   { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
567 }
568 \int_if_zero:nT \l_tmpa_int
569 {
570   \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
571   \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
572 }
573 \seq_put_right:Ne \l_@@_notes_labels_seq
574 {
575   \tl_if_novalue:nTF { #1 }
576   {
577     \@@_notes_format:n
578     {
579       \int_eval:n

```

```

580         {
581             \int_if_zero:nTF \l_tmpa_int
582                 \c@tabularnote
583                 \l_tmpa_int
584         }
585     }
586 }
587 { #1 }
588 }
589 \peek_meaning:Nf \tabularnote
590 {

```

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`.

```

591     \hbox_set:Nn \l_tmpa_box
592     {

```

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

```

593         \@@_notes_label_in_tabular:n
594         {
595             \seq_use:Nnnn
596             \l_@@_notes_labels_seq { , } { , } { , }
597         }
598     }

```

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

```

599     \int_gdecr:N \c@tabularnote
600     \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.

```

601     \int_gincr:N \g_@@_tabularnote_int
602     \refstepcounter { tabularnote }
603     \int_compare:nNnT \l_tmpa_int = \c@tabularnote
604     { \int_gincr:N \c@tabularnote }
605     \seq_clear:N \l_@@_notes_labels_seq
606     \bool_lazy_or:nnTF
607     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { c } }
608     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { r } }
609     {
610         \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).

```

611         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
612     }
613     { \box_use:N \l_tmpa_box }
614 }
615 }

```

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`.

```

616 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
617 {
618     \bool_if:Nf \g_@@_caption_finished_bool
619     {

```

```

620     \int_compare:nNnT \c@tabulernote = \g_@@_notes_caption_int
621     { \int_gzero:N \c@tabulernote }

```

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`!

```

622     \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
623     { \@@_error:n { Identical-notes-in-caption } }
624   }
625   {

```

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

```

626     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
627     {

```

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 `\tabulernote` in the caption.

```

628         \bool_gset_true:N \g_@@_caption_finished_bool
629         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabulernote
630         \int_gzero:N \c@tabulernote
631       }
632     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
633   }

```

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!

```

634     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabulernote }
635     \seq_put_right:Ne \l_@@_notes_labels_seq
636     {
637       \tl_if_novalue:nTF { #1 }
638       { \@@_notes_format:n { \int_use:N \c@tabulernote } }
639       { #1 }
640     }
641     \peek_meaning:NF \tabulernote
642     {
643       \@@_notes_label_in_tabular:n
644       { \seq_use:Nmn \l_@@_notes_labels_seq { , } { , } { , } }
645       \seq_clear:N \l_@@_notes_labels_seq
646     }
647   }

648 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
649 { \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.

```

650 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
651 {
652   \begin { pgfscope }
653   \pgfset
654   {
655     inner~sep = \c_zero_dim ,
656     minimum~size = \c_zero_dim
657   }
658   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
659   \pgfnode
660   { rectangle }

```

```

661     { center }
662     {
663       \vbox_to_ht:nn
664         { \dim_abs:n { #5 - #3 } }
665         {
666           \vfill
667           \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
668         }
669     }
670     { #1 }
671     { }
672 \end { pgfscope }
673 }

```

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.

```

674 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
675 {
676   \begin { pgfscope }
677   \pgfset
678     {
679       inner~sep = \c_zero_dim ,
680       minimum~size = \c_zero_dim
681     }
682   \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
683   \pgfpointdiff { #3 } { #2 }
684   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
685   \pgfnode
686     { rectangle }
687     { center }
688     {
689       \vbox_to_ht:nn
690         { \dim_abs:n \l_tmpb_dim }
691         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
692     }
693     { #1 }
694     { }
695   \end { pgfscope }
696 }

```

## 7 The options

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

```

697 \tl_new:N \l_@@_caption_tl
698 \tl_new:N \l_@@_short_caption_tl
699 \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).

```

700 \bool_new:N \l_@@_caption_above_bool

```

By default, the commands `\cellcolor` and `\rowcolor` are available for the user in the cells of the tabular (the user may use the commands provided by `\colortbl`). However, if the key `color-inside` is used, these commands are available.

```

701 \bool_new:N \l_@@_color_inside_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`.

```
702 \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`).

```
703 \dim_new:N \l_@@_cell_space_top_limit_dim
704 \dim_new:N \l_@@_cell_space_bottom_limit_dim
```

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

```
705 \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.

```
706 \dim_new:N \l_@@_xdots_inter_dim
707 \hook_gput_code:nnn { begindocument } { . }
708 { \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).

```
709 \dim_new:N \l_@@_xdots_shorten_start_dim
710 \dim_new:N \l_@@_xdots_shorten_end_dim
711 \hook_gput_code:nnn { begindocument } { . }
712 {
713   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
714   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
715 }
```

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.

```
716 \dim_new:N \l_@@_xdots_radius_dim
717 \hook_gput_code:nnn { begindocument } { . }
718 { \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.

```
719 \tl_new:N \l_@@_xdots_line_style_tl
720 \tl_const:Nn \c_@@_standard_tl { standard }
721 \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`.

```
722 \bool_new:N \l_@@_light_syntax_bool
723 \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).

```
724 \tl_new:N \l_@@_baseline_tl
725 \tl_set:Nn \l_@@_baseline_tl { c }
```

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

```
726 \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`).

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

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

```
728 \bool_new:N \l_@@_parallelize_diags_bool
729 \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.

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

```
731 \dim_new:N \l_@@_notes_above_space_dim
732 \hook_gput_code:nnn { begindocument } { . }
733 { \dim_set:Nn \l_@@_notes_above_space_dim { 1 mm } }
```

We use a hook only by security in case `revtex4-1` is used (even though it is obsolete).

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.).

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

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

```
735 \cs_new_protected:Npn \@@_reset_arraystretch:
736 { \cs_set_nopar:Npn \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).

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

The following boolean corresponds to the key `create-cell-nodes` of the keyword `\CodeBefore`.

```
738 \bool_new:N \g_@@_recreate_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.

```
739 \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”.

```
740 \bool_new:N \l_@@_medium_nodes_bool
741 \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).

```
742 \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`).

```
743 \dim_new:N \l_@@_left_margin_dim
744 \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`.

```
745 \dim_new:N \l_@@_extra_left_margin_dim
746 \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.

```
747 \tl_new:N \l_@@_end_of_row_tl
748 \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 “:”.

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

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

```
750 \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 acheive 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.

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

```
752 \keys_define:nn { nicematrix / xdots }
753 {
754   shorten-start .code:n =
755     \hook_gput_code:nnn { begindocument } { . }
756     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
757   shorten-end .code:n =
758     \hook_gput_code:nnn { begindocument } { . }
759     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
760   shorten-start .value_required:n = true ,
761   shorten-end .value_required:n = true ,
762   shorten .code:n =
763     \hook_gput_code:nnn { begindocument } { . }
764     {
765       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
766       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
767     } ,
768   shorten .value_required:n = true ,
769   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
770   horizontal-labels .default:n = true ,
771   line-style .code:n =
772     {
773       \bool_lazy_or:nnTF
774         { \cs_if_exist_p:N \tikzpicture }
775         { \str_if_eq_p:mn { #1 } { standard } }
776         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
777         { \@@_error:n { bad-option-for-line-style } }
778     } ,
```

```

779 line-style .value_required:n = true ,
780 color .tl_set:N = \l_@@_xdots_color_tl ,
781 color .value_required:n = true ,
782 radius .code:n =
783   \hook_gput_code:nnn { begindocument } { . }
784   { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
785 radius .value_required:n = true ,
786 inter .code:n =
787   \hook_gput_code:nnn { begindocument } { . }
788   { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
789 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}`.

```

790 down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
791 up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
792 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).

```

793 draw-first .code:n = \prg_do_nothing: ,
794 unknown .code:n = \@@_error:n { Unknown~key~for~xdots }
795 }

```

```

796 \keys_define:nn { nicematrix / rules }
797 {
798   color .tl_set:N = \l_@@_rules_color_tl ,
799   color .value_required:n = true ,
800   width .dim_set:N = \arrayrulewidth ,
801   width .value_required:n = true ,
802   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
803 }

```

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

```

804 \keys_define:nn { nicematrix / Global }
805 {
806   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
807   ampersand-in-blocks .default:n = true ,
808   &-in-blocks .meta:n = ampersand-in-blocks ,
809   no-cell-nodes .code:n =
810     \cs_set_protected:Npn \@@_node_for_cell:
811     { \box_use_drop:N \l_@@_cell_box } ,
812   no-cell-nodes .value_forbidden:n = true ,
813   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
814   rounded-corners .default:n = 4 pt ,
815   custom-line .code:n = \@@_custom_line:n { #1 } ,
816   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
817   rules .value_required:n = true ,
818   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
819   standard-cline .default:n = true ,
820   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
821   cell-space-top-limit .value_required:n = true ,
822   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
823   cell-space-bottom-limit .value_required:n = true ,
824   cell-space-limits .meta:n =
825     {
826       cell-space-top-limit = #1 ,
827       cell-space-bottom-limit = #1 ,
828     } ,

```

```

829 cell-space-limits .value_required:n = true ,
830 xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
831 light-syntax .code:n =
832   \bool_set_true:N \l_@@_light_syntax_bool
833   \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
834 light-syntax .value_forbidden:n = true ,
835 light-syntax-expanded .code:n =
836   \bool_set_true:N \l_@@_light_syntax_bool
837   \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
838 light-syntax-expanded .value_forbidden:n = true ,
839 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
840 end-of-row .value_required:n = true ,
841 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
842 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
843 last-row .int_set:N = \l_@@_last_row_int ,
844 last-row .default:n = -1 ,
845 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
846 code-for-first-col .value_required:n = true ,
847 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
848 code-for-last-col .value_required:n = true ,
849 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
850 code-for-first-row .value_required:n = true ,
851 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
852 code-for-last-row .value_required:n = true ,
853 hlines .clist_set:N = \l_@@_hlines_clist ,
854 vlines .clist_set:N = \l_@@_vlines_clist ,
855 hlines .default:n = all ,
856 vlines .default:n = all ,
857 vlines-in-sub-matrix .code:n =
858   {
859     \tl_if_single_token:nTF { #1 }
860     {
861       \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
862       { \@@_error:nn { Forbidden-letter } { #1 } }

```

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

```

863     { \cs_set_eq:cN { @@ _ #1 } \@@_make_preamble_vlism:n }
864   }
865   { \@@_error:n { One-letter-allowed } }
866 } ,
867 vlines-in-sub-matrix .value_required:n = true ,
868 hvlines .code:n =
869   {
870     \bool_set_true:N \l_@@_hvlines_bool
871     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
872     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
873   } ,
874 hvlines-except-borders .code:n =
875   {
876     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
877     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
878     \bool_set_true:N \l_@@_hvlines_bool
879     \bool_set_true:N \l_@@_except_borders_bool
880   } ,
881 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.

```

882 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
883 renew-dots .value_forbidden:n = true ,
884 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
885 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
886 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,

```

```

887 create-extra-nodes .meta:n =
888   { create-medium-nodes , create-large-nodes } ,
889 left-margin .dim_set:N = \l_@@_left_margin_dim ,
890 left-margin .default:n = \arraycolsep ,
891 right-margin .dim_set:N = \l_@@_right_margin_dim ,
892 right-margin .default:n = \arraycolsep ,
893 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
894 margin .default:n = \arraycolsep ,
895 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
896 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
897 extra-margin .meta:n =
898   { extra-left-margin = #1 , extra-right-margin = #1 } ,
899 extra-margin .value_required:n = true ,
900 respect-arraystretch .code:n =
901   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
902 respect-arraystretch .value_forbidden:n = true ,
903 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
904 pgf-node-code .value_required:n = true
905 }

```

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

```

906 \keys_define:nn { nicematrix / environments }
907 {
908   corners .clist_set:N = \l_@@_corners_clist ,
909   corners .default:n = { NW , SW , NE , SE } ,
910   code-before .code:n =
911     {
912       \tl_if_empty:nF { #1 }
913       {
914         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
915         \bool_set_true:N \l_@@_code_before_bool
916       }
917     } ,
918   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}`.

```

919   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
920   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
921   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
922   baseline .tl_set:N = \l_@@_baseline_tl ,
923   baseline .value_required:n = true ,
924   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)`.

```

925   \str_if_eq:eeTF { #1 } { auto }
926   { \bool_set_true:N \l_@@_auto_columns_width_bool }
927   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
928   columns-width .value_required:n = true ,
929   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.

```

930   \legacy_if:nF { measuring@ }
931   {
932     \str_set:Ne \l_tmpa_str { #1 }
933     \seq_if_in:NoTF \g_@@_names_seq \l_tmpa_str
934     { \@@_error:nn { Duplicate~name } { #1 } }
935     { \seq_gput_left:No \g_@@_names_seq \l_tmpa_str }
936     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
937   } ,

```

```

938     name .value_required:n = true ,
939     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
940     code-after .value_required:n = true ,
941     color-inside .code:n =
942       \bool_set_true:N \l_@@_color_inside_bool
943       \bool_set_true:N \l_@@_code_before_bool ,
944     color-inside .value_forbidden:n = true ,
945     colortbl-like .meta:n = color-inside
946   }
947 \keys_define:nn { nicematrix / notes }
948   {
949     para .bool_set:N = \l_@@_notes_para_bool ,
950     para .default:n = true ,
951     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
952     code-before .value_required:n = true ,
953     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
954     code-after .value_required:n = true ,
955     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
956     bottomrule .default:n = true ,
957     style .cs_set:Np = \@@_notes_style:n #1 ,
958     style .value_required:n = true ,
959     label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
960     label-in-tabular .value_required:n = true ,
961     label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
962     label-in-list .value_required:n = true ,
963     enumitem-keys .code:n =
964       {
965         \hook_gput_code:nnn { begindocument } { . }
966         {
967           \IfPackageLoadedT { enumitem }
968             { \setlist* [ tabularnotes ] { #1 } }
969         }
970       } ,
971     enumitem-keys .value_required:n = true ,
972     enumitem-keys-para .code:n =
973       {
974         \hook_gput_code:nnn { begindocument } { . }
975         {
976           \IfPackageLoadedT { enumitem }
977             { \setlist* [ tabularnotes* ] { #1 } }
978         }
979       } ,
980     enumitem-keys-para .value_required:n = true ,
981     detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
982     detect-duplicates .default:n = true ,
983     unknown .code:n = \@@_error:n { Unknown~key~for~notes }
984   }
985 \keys_define:nn { nicematrix / delimiters }
986   {
987     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
988     max-width .default:n = true ,
989     color .tl_set:N = \l_@@_delimiters_color_tl ,
990     color .value_required:n = true ,
991   }

```

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

```

992 \keys_define:nn { nicematrix }
993   {
994     NiceMatrixOptions .inherit:n =
995       { nicematrix / Global } ,
996     NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,

```

```

997 NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
998 NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
999 NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1000 SubMatrix / rules .inherit:n = nicematrix / rules ,
1001 CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1002 CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1003 CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1004 NiceMatrix .inherit:n =
1005   {
1006     nicematrix / Global ,
1007     nicematrix / environments ,
1008   } ,
1009 NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1010 NiceMatrix / rules .inherit:n = nicematrix / rules ,
1011 NiceTabular .inherit:n =
1012   {
1013     nicematrix / Global ,
1014     nicematrix / environments
1015   } ,
1016 NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1017 NiceTabular / rules .inherit:n = nicematrix / rules ,
1018 NiceTabular / notes .inherit:n = nicematrix / notes ,
1019 NiceArray .inherit:n =
1020   {
1021     nicematrix / Global ,
1022     nicematrix / environments ,
1023   } ,
1024 NiceArray / xdots .inherit:n = nicematrix / xdots ,
1025 NiceArray / rules .inherit:n = nicematrix / rules ,
1026 pNiceArray .inherit:n =
1027   {
1028     nicematrix / Global ,
1029     nicematrix / environments ,
1030   } ,
1031 pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1032 pNiceArray / rules .inherit:n = nicematrix / rules ,
1033 }

```

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

```

1034 \keys_define:nn { nicematrix / NiceMatrixOptions }
1035   {
1036     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1037     delimiters / color .value_required:n = true ,
1038     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1039     delimiters / max-width .default:n = true ,
1040     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1041     delimiters .value_required:n = true ,
1042     width .dim_set:N = \l_@@_width_dim ,
1043     width .value_required:n = true ,
1044     last-col .code:n =
1045       \tl_if_empty:nF { #1 }
1046         { \@@_error:n { last-col~non-empty~for~NiceMatrixOptions } }
1047       \int_zero:N \l_@@_last_col_int ,
1048     small .bool_set:N = \l_@@_small_bool ,
1049     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.

```

1050     renew-matrix .code:n = \@@_renew_matrix: ,
1051     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}`.

```
1052     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.

```
1053     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)`.

```
1054     \str_if_eq:eeTF { #1 } { auto }
1055     { \@@_error:n { Option~auto~for~columns~width } }
1056     { \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.

```
1057     allow-duplicate-names .code:n =
1058     \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
1059     allow-duplicate-names .value_forbidden:n = true ,
1060     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1061     notes .value_required:n = true ,
1062     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1063     sub-matrix .value_required:n = true ,
1064     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1065     matrix / columns-type .value_required:n = true ,
1066     caption-above .bool_set:N = \l_@@_caption_above_bool ,
1067     caption-above .default:n = true ,
1068     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1069 } ,
```

`\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.

```
1070 \NewDocumentCommand \NiceMatrixOptions { m }
1071 { \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.

```
1072 \keys_define:nn { nicematrix / NiceMatrix }
1073 {
1074     last-col .code:n = \tl_if_empty:nTF { #1 }
1075     {
1076         \bool_set_true:N \l_@@_last_col_without_value_bool
1077         \int_set:Nn \l_@@_last_col_int { -1 }
1078     }
1079     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1080     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1081     columns-type .value_required:n = true ,
1082     l .meta:n = { columns-type = l } ,
1083     r .meta:n = { columns-type = r } ,
1084     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1085     delimiters / color .value_required:n = true ,
1086     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1087     delimiters / max-width .default:n = true ,
1088     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1089     delimiters .value_required:n = true ,
1090     small .bool_set:N = \l_@@_small_bool ,
1091     small .value_forbidden:n = true ,
1092     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1093 }
```

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

```
1094 \keys_define:nn { nicematrix / NiceArray }
1095 {
```

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.

```
1096     small .bool_set:N = \l_@@_small_bool ,
1097     small .value_forbidden:n = true ,
1098     last-col .code:n = \tl_if_empty:nF { #1 }
1099         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1100         \int_zero:N \l_@@_last_col_int ,
1101     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1102     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1103     unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
1104 }
```

```
1105 \keys_define:nn { nicematrix / pNiceArray }
1106 {
1107     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1108     last-col .code:n = \tl_if_empty:nF { #1 }
1109         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1110         \int_zero:N \l_@@_last_col_int ,
1111     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1112     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1113     delimiters / color .value_required:n = true ,
1114     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1115     delimiters / max-width .default:n = true ,
1116     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1117     delimiters .value_required:n = true ,
1118     small .bool_set:N = \l_@@_small_bool ,
1119     small .value_forbidden:n = true ,
1120     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1121     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1122     unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1123 }
```

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

```
1124 \keys_define:nn { nicematrix / NiceTabular }
1125 {
```

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.

```
1126     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1127         \bool_set_true:N \l_@@_width_used_bool ,
1128     width .value_required:n = true ,
1129     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1130     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1131     tabularnote .value_required:n = true ,
1132     caption .tl_set:N = \l_@@_caption_tl ,
1133     caption .value_required:n = true ,
1134     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1135     short-caption .value_required:n = true ,
1136     label .tl_set:N = \l_@@_label_tl ,
1137     label .value_required:n = true ,
1138     last-col .code:n = \tl_if_empty:nF { #1 }
1139         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1140         \int_zero:N \l_@@_last_col_int ,
1141     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1142     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1143     unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
1144 }
```

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
```

```
1145 \keys_define:nn { nicematrix / CodeAfter }
1146 {
1147   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1148   delimiters / color .value_required:n = true ,
1149   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1150   rules .value_required:n = true ,
1151   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1152   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1153   sub-matrix .value_required:n = true ,
1154   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1155 }
```

## 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}`).

```
1156 \cs_new_protected:Npn \@@_cell_begin:
1157 {
```

`\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.

```
1158   \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).

```
1159   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1160   \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.

```
1161   \int_compare:nNnT \c@jCol = \c_one_int
1162     { \int_compare:nNnT \l_@@_first_col_int = \c_one_int \@@_begin_of_row: }
```

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:`.

```
1163   \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1164   \@@_tuning_not_tabular_begin:
```

```
1165   \@@_tuning_first_row:
```

```
1166   \@@_tuning_last_row:
```

```
1167   \g_@@_row_style_tl
```

```
1168 }
```

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_compare:nNnT \c@jCol > 0
    {
      \l_@@_code_for_first_row_tl
      \xglobal \colorlet { nicematrix-first-row } { . }
    }
  }
}

```

We will use a version a little more efficient.

```

1169 \cs_new_protected:Npn \@@_tuning_first_row:
1170 {
1171   \if_int_compare:w \c@iRow = \c_zero_int
1172     \if_int_compare:w \c@jCol > \c_zero_int
1173       \l_@@_code_for_first_row_tl
1174       \xglobal \colorlet { nicematrix-first-row } { . }
1175     \fi:
1176   \fi:
1177 }

```

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:nNnT \c@iRow = \l_@@_last_row_int
  {
    \l_@@_code_for_last_row_tl
    \xglobal \colorlet { nicematrix-last-row } { . }
  }
}

```

We will use a version a little more efficient.

```

1178 \cs_new_protected:Npn \@@_tuning_last_row:
1179 {
1180   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1181     \l_@@_code_for_last_row_tl
1182     \xglobal \colorlet { nicematrix-last-row } { . }
1183   \fi:
1184 }

```

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

```

1185 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1186 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1187 {
1188   \c_math_toggle_token

```

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

```

1189   \@@_tuning_key_small:
1190 }
1191 \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.

```

1192 \cs_new_protected:Npn \@@_begin_of_row:
1193 {
1194   \int_gincr:N \c@iRow
1195   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim

```

```

1196 \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1197 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1198 \pgfpicture
1199 \pgfrememberpicturepositiononpagetrue
1200 \pgfcoordinate
1201 { \@@_env: - row - \int_use:N \c@iRow - base }
1202 { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1203 \str_if_empty:NF \l_@@_name_str
1204 {
1205 \pgfnodealias
1206 { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1207 { \@@_env: - row - \int_use:N \c@iRow - base }
1208 }
1209 \endpgfpicture
1210 }

```

Remark: If the key `recreate-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 informations 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.

```

1211 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1212 {
1213 \int_if_zero:nTF \c@iRow
1214 {
1215 \dim_compare:nNnT { \box_dp:N \l_@@_cell_box } > \g_@@_dp_row_zero_dim
1216 { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1217 \dim_compare:nNnT { \box_ht:N \l_@@_cell_box } > \g_@@_ht_row_zero_dim
1218 { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1219 }
1220 {
1221 \int_compare:nNnT \c@iRow = \c_one_int
1222 {
1223 \dim_compare:nNnT { \box_ht:N \l_@@_cell_box } > \g_@@_ht_row_zero_dim
1224 { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1225 }
1226 }
1227 }
1228 \cs_new_protected:Npn \@@_rotate_cell_box:
1229 {
1230 \box_rotate:Nn \l_@@_cell_box { 90 }
1231 \bool_if:NTF \g_@@_rotate_c_bool
1232 {
1233 \hbox_set:Nn \l_@@_cell_box
1234 {
1235 \c_math_toggle_token
1236 \vcenter { \box_use:N \l_@@_cell_box }
1237 \c_math_toggle_token
1238 }
1239 }
1240 {
1241 \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1242 {
1243 \vbox_set_top:Nn \l_@@_cell_box
1244 {
1245 \vbox_to_zero:n { }
1246 \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1247 \box_use:N \l_@@_cell_box
1248 }
1249 }
1250 }
1251 \bool_gset_false:N \g_@@_rotate_bool

```

```

1252   \bool_gset_false:N \g_@@_rotate_c_bool
1253 }
1254 \cs_new_protected:Npn \@@_adjust_size_box:
1255 {
1256   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1257   {
1258     \box_set_wd:Nn \l_@@_cell_box
1259     { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1260     \dim_gzero:N \g_@@_blocks_wd_dim
1261   }
1262   \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1263   {
1264     \box_set_dp:Nn \l_@@_cell_box
1265     { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1266     \dim_gzero:N \g_@@_blocks_dp_dim
1267   }
1268   \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1269   {
1270     \box_set_ht:Nn \l_@@_cell_box
1271     { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1272     \dim_gzero:N \g_@@_blocks_ht_dim
1273   }
1274 }
1275 \cs_new_protected:Npn \@@_cell_end:
1276 {

```

The following command is nullified in the tabulars.

```

1277   \@@_tuning_not_tabular_end:
1278   \hbox_set_end:
1279   \@@_cell_end_i:
1280 }
1281 \cs_new_protected:Npn \@@_cell_end_i:
1282 {

```

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.

```

1283   \g_@@_cell_after_hook_tl
1284   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1285   \@@_adjust_size_box:
1286   \box_set_ht:Nn \l_@@_cell_box
1287   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1288   \box_set_dp:Nn \l_@@_cell_box
1289   { \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”).

```

1290   \@@_update_max_cell_width:

```

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

```

1291   \@@_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 technic:

- 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.

```

1292   \bool_if:NTF \g_@@_empty_cell_bool
1293     { \box_use_drop:N \l_@@_cell_box }
1294     {
1295       \bool_if:NTF \g_@@_not_empty_cell_bool
1296         \@@_node_for_cell:
1297         {
1298           \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1299             \@@_node_for_cell:
1300             { \box_use_drop:N \l_@@_cell_box }
1301         }
1302     }
1303   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
1304     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1305   \bool_gset_false:N \g_@@_empty_cell_bool
1306   \bool_gset_false:N \g_@@_not_empty_cell_bool
1307 }

```

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

```

1308 \cs_new_protected:Npn \@@_update_max_cell_width:
1309   {
1310     \dim_gset:Nn \g_@@_max_cell_width_dim
1311     { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1312   }

```

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`).

```

1313 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1314   {
1315     \@@_math_toggle:
1316     \hbox_set_end:
1317     \bool_if:NF \g_@@_rotate_bool
1318       {
1319         \hbox_set:Nn \l_@@_cell_box
1320         {
1321           \makebox [ \l_@@_col_width_dim ] [ s ]
1322           { \hbox_unpack_drop:N \l_@@_cell_box }
1323         }
1324       }
1325     \@@_cell_end_i:
1326   }

```

```

1327 \pgfset
1328   {
1329     nicematrix / cell-node /.style =
1330     {
1331       inner~sep = \c_zero_dim ,
1332       minimum~width = \c_zero_dim
1333     }
1334   }

```

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

```

1335 \cs_new_protected:Npn \@@_node_for_cell:
1336 {
1337   \pgfpicture
1338   \pgfsetbaseline \c_zero_dim
1339   \pgfrememberpicturepositiononpagetrue
1340   \pgfset { nicematrix / cell-node }
1341   \pgfnode
1342     { rectangle }
1343     { base }
1344   {

```

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

```

1345     \set@color
1346     \box_use_drop:N \l_@@_cell_box
1347   }
1348   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1349   { \l_@@_pgf_node_code_tl }
1350   \str_if_empty:NF \l_@@_name_str
1351   {
1352     \pgfnodealias
1353       { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1354       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1355   }
1356   \endpgfpicture
1357 }

```

As its name says, the following command is a patch for the command `\@@_node_for_cell:`. This patch will be appended on the left of `\@@_node_for_the_cell:` when the construction of the cell nodes (of the form  $(i-j)$ ) in the `\CodeBefore` is required.

```

1358 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1359 {
1360   \cs_new_protected:Npn \@@_patch_node_for_cell:
1361   {
1362     \hbox_set:Nn \l_@@_cell_box
1363     {
1364       \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1365       \hbox_overlap_left:n
1366       {
1367         \pgfsys@markposition
1368         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }

```

I don't know why the following adjustment is needed when the compilation is done with XeLaTeX or with the classical way `latex`, `dvips`, `ps2pdf` (or Adobe Distiller). However, it seems to work.

```

1369         #1
1370       }
1371       \box_use:N \l_@@_cell_box
1372       \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1373       \hbox_overlap_left:n
1374       {
1375         \pgfsys@markposition
1376         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1377         #1
1378       }
1379     }
1380   }
1381 }

```

We have no explanation for the different behaviour between the TeX engines...

```

1382 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1383 {

```



```

1384 \@@_patch_node_for_cell:n
1385   { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1386 }
1387 { \@@_patch_node_for_cell:n { } }

```

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

For example, for the following matrix,

```

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

```

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

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

```

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

```

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

```

1388 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1389   {
1390     \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce
1391     { g_@@_#2 _ lines _ tl }
1392     {
1393       \use:c { @@ _ draw _ #2 : nnn }
1394       { \int_use:N \c@iRow }
1395       { \int_use:N \c@jCol }
1396       { \exp_not:n { #3 } }
1397     }
1398   }

```

```

1399 \cs_generate_variant:Nn \@@_array:n { o }
1400 \cs_new_protected:Npn \@@_array:n
1401   {
1402   % \begin{macrocode}
1403   \dim_set:Nn \col@sep
1404     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1405   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1406     { \cs_set_nopar:Npn \@halignto { } }
1407     { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1408 \@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)`.

```

1409 [ \str_if_eq:eeTF \l_@@_baseline_tl c c t ]
1410 ]

```

We keep in memory the standard version of `\ialign` because we will redefine `\ialign` in the environment `{NiceArrayWithDelims}` but restore the standard version for use in the cells of the array. However, since version 2.6a (version for the Tagging Project), `array` uses `\ar@ialign` instead of `\ialign`. In that case, of course, you do a saving of `\ar@ialign`.

```

1411 \bool_if:NTF \c_@@_tagging_array_bool
1412   { \cs_set_eq:NN \@@_old_ar@ialign: \ar@ialign }
1413   { \cs_set_eq:NN \@@_old_ialign: \ialign }

```

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

```

1414 \cs_new_protected:Npn \@@_create_row_node:
1415 {
1416   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1417   {
1418     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1419     \@@_create_row_node_i:
1420   }
1421 }

1422 \cs_new_protected:Npn \@@_create_row_node_i:
1423 {

```

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

```

1424   \hbox
1425   {
1426     \bool_if:NT \l_@@_code_before_bool
1427     {
1428       \vtop
1429       {
1430         \skip_vertical:N 0.5\arrayrulewidth
1431         \pgfsys@markposition
1432         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1433         \skip_vertical:N -0.5\arrayrulewidth
1434       }
1435     }
1436     \pgfpicture
1437     \pgfrememberpicturepositiononpagetrue
1438     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1439     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1440     \str_if_empty:NF \l_@@_name_str
1441     {
1442       \pgfnodealias
1443       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1444       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1445     }
1446     \endpgfpicture
1447   }
1448 }

```

The following must *not* be protected because it begins with `\noalign`.

```

1449 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }

1450 \cs_new_protected:Npn \@@_everycr_i:
1451 {
1452   \bool_if:NT \c_@@_testphase_table_bool
1453   {
1454     \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1455     \tbl_update_cell_data_for_next_row:
1456   }
1457   \int_gzero:N \c@jCol
1458   \bool_gset_false:N \g_@@_after_col_zero_bool
1459   \bool_if:NF \g_@@_row_of_col_done_bool
1460   {
1461     \@@_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).

```

1462     \clist_if_empty:NF \l_@@_hlines_clist
1463     {
1464       \str_if_eq:eeF \l_@@_hlines_clist { all }
1465       {
1466         \clist_if_in:NeT

```

```

1467         \l_@@_hlines_clist
1468         { \int_eval:n { \c@iRow + 1 } }
1469     }
1470 {

```

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.

```

1471         \int_compare:nNnT \c@iRow > { -1 }
1472         {
1473             \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1474                 { \hrule height \arrayrulewidth width \c_zero_dim }
1475         }
1476     }
1477 }
1478 }
1479 }

```

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

```

1480 \cs_set_protected:Npn \@@_renew_dots:
1481 {
1482     \cs_set_eq:NN \ldots \@@_Ldots
1483     \cs_set_eq:NN \cdots \@@_Cdots
1484     \cs_set_eq:NN \vdots \@@_Vdots
1485     \cs_set_eq:NN \ddots \@@_Ddots
1486     \cs_set_eq:NN \iddots \@@_Iddots
1487     \cs_set_eq:NN \dots \@@_Ldots
1488     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1489 }
1490 \cs_new_protected:Npn \@@_test_color_inside:
1491 {
1492     \bool_if:NF \l_@@_color_inside_bool
1493     {

```

We will issue an error only during the first run.

```

1494         \bool_if:NF \g_@@_aux_found_bool
1495         { \@@_error:n { without~color~inside } }
1496     }
1497 }

```

```

1498 \cs_new_protected:Npn \@@_redefine_everycr:
1499 { \everycr { \@@_everycr: } }
1500 \hook_gput_code:nnn { begindocument } { . }
1501 {
1502     \IfPackageLoadedT { colortbl }
1503     {
1504         \cs_set_protected:Npn \@@_redefine_everycr:
1505         {
1506             \CT@everycr
1507             {
1508                 \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1509                 \@@_everycr:
1510             }
1511         }
1512     }
1513 }

```

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 <sup>4</sup>.

```

1514 \hook_gput_code:nnn { begindocument } { . }
1515 {
1516   \IfPackageLoadedTF { booktabs }
1517   {
1518     \cs_new_protected:Npn \@@_patch_booktabs:
1519     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1520   }
1521   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1522 }

```

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`<sup>5</sup> 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`.

```

1523 \cs_new_protected:Npn \@@_some_initialization:
1524 {
1525   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1526   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1527   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1528   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1529   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1530   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1531 }

```

```

1532 \cs_new_protected:Npn \@@_pre_array_ii:
1533 {

```

The number of letters `X` in the preamble of the array.

```

1534   \int_gzero:N \g_@@_total_X_weight_int
1535   \@@_expand_clist:N \l_@@_hlines_clist
1536   \@@_expand_clist:N \l_@@_vlines_clist
1537   \@@_patch_booktabs:
1538   \box_clear_new:N \l_@@_cell_box
1539   \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}`).

```

1540   \bool_if:NT \l_@@_small_bool
1541   {
1542     \cs_set_nopar:Npn \arraystretch { 0.47 }
1543     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1544     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1545   }

1546   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1547   {
1548     \tl_put_right:Nn \@@_begin_of_row:

```

---

<sup>4</sup>cf. `\nicematrix@redefine@check@rerun`

<sup>5</sup>The option `small` of `nicematrix` changes (among others) the value of `\arraystretch`. This is done, of course, before the call of `{array}`.

```

1549     {
1550         \pgfsys@markposition
1551         { \@@_env: - row - \int_use:N \c@iRow - base }
1552     }
1553 }

```

The environment `{array}` uses internally the command `\ialign`. We change the definition of `\ialign` for several reasons. In particular, `\ialign` sets `\everycr` to `{ }` and we *need* to have to change the value of `\everycr`.

```

1554     \bool_if:NTF \c_@@_tagging_array_bool
1555     {
1556         \cs_set_nopar:Npn \ar@ialign
1557         {
1558             \bool_if:NT \c_@@_testphase_table_bool \tbl_init_cell_data_for_table:
1559             \@@_redefine_everycr:
1560             \dim_zero:N \tabskip
1561             \@@_some_initialization:

```

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`.

```

1562             \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
1563             \halign
1564         }
1565     }

```

The following part will be deleted when we will delete the boolean `\c_@@_tagging_array_bool` (when we consider the version 2.6a of `array` is required).

```

1566     {
1567         \cs_set_nopar:Npn \ialign
1568         {
1569             \@@_redefine_everycr:
1570             \dim_zero:N \tabskip
1571             \@@_some_initialization:
1572             \cs_set_eq:NN \ialign \@@_old_ialign:
1573             \halign
1574         }
1575     }

```

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).

```

1576     \cs_set_eq:NN \@@_old_ldots \ldots
1577     \cs_set_eq:NN \@@_old_cdots \cdots
1578     \cs_set_eq:NN \@@_old_vdots \vdots
1579     \cs_set_eq:NN \@@_old_ddots \ddots
1580     \cs_set_eq:NN \@@_old_iddots \iddots
1581     \bool_if:NTF \l_@@_standard_cline_bool
1582     { \cs_set_eq:NN \cline \@@_standard_cline }
1583     { \cs_set_eq:NN \cline \@@_cline }
1584     \cs_set_eq:NN \Ldots \@@_Ldots
1585     \cs_set_eq:NN \Cdots \@@_Cdots
1586     \cs_set_eq:NN \Vdots \@@_Vdots
1587     \cs_set_eq:NN \Ddots \@@_Ddots
1588     \cs_set_eq:NN \Iddots \@@_Iddots
1589     \cs_set_eq:NN \Hline \@@_Hline:
1590     \cs_set_eq:NN \Hspace \@@_Hspace:
1591     \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1592     \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1593     \cs_set_eq:NN \Block \@@_Block:
1594     \cs_set_eq:NN \rotate \@@_rotate:
1595     \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1596     \cs_set_eq:NN \dotfill \@@_dotfill:
1597     \cs_set_eq:NN \CodeAfter \@@_CodeAfter:

```

```

1598 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1599 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1600 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1601 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1602   { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1603 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1604 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1605 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1606 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1607 \int_compare:nNnT \l_@@_first_row_int > \c_zero_int
1608   { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1609 \int_compare:nNnT \l_@@_last_row_int < \c_zero_int
1610   { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1611 \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:`.

```

1612 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1613 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1614   { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1615 \@@_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!).

```

1616 \tl_if_exist:NT \l_@@_note_in_caption_tl
1617   {
1618     \tl_if_empty:NF \l_@@_note_in_caption_tl
1619     {
1620       \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1621       \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1622     }
1623   }

```

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).

```

1624 \seq_gclear:N \g_@@_multicolumn_cells_seq
1625 \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).

```

1626 \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 or rows excepted the last row (if `\l_@@_last_row_bool` has been raised with the option `last-row`).

```

1627 \int_gzero_new: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.

```

1628 \int_gzero_new:N \g_@@_col_total_int
1629 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1630 \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.

```

1631 \tl_gclear_new:N \g_@@_Cdots_lines_tl

```

```

1632 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1633 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1634 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1635 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1636 \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

1637 \tl_gclear:N \g_nicematrix_code_before_tl
1638 \tl_gclear:N \g_@@_pre_code_before_tl
1639 }

```

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

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

```

1640 \cs_new_protected:Npn \@@_pre_array:
1641 {
1642   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1643   \int_gzero_new:N \c@iRow
1644   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1645   \int_gzero_new:N \c@jCol

```

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).

```

1646 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1647 {
1648   \bool_set_true:N \l_@@_last_row_without_value_bool
1649   \bool_if:NT \g_@@_aux_found_bool
1650     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1651 }
1652 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1653 {
1654   \bool_if:NT \g_@@_aux_found_bool
1655     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1656 }

```

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”.

```

1657 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1658 {
1659   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1660     {
1661       \dim_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1662         { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1663       \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1664         { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1665     }
1666 }

1667 \seq_gclear:N \g_@@_cols_vlism_seq
1668 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1669 \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.

```

1670 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```
1671 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1672 \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.

```
1673 \int_gset:Nn \g_@@_last_row_node_int { -2 }
```

The value `-2` is important.

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

```
1674 \@@_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.

```
1675 \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]`).

```
1676 \dim_zero_new:N \l_@@_left_delim_dim
1677 \dim_zero_new:N \l_@@_right_delim_dim
1678 \bool_if:NTF \g_@@_delims_bool
1679 {
```

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

```
1680 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1681 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1682 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1683 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1684 }
1685 {
1686 \dim_gset:Nn \l_@@_left_delim_dim
1687 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1688 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1689 }
```

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).

```
1690 \hbox_set:Nw \l_@@_the_array_box
1691 \bool_if:NT \c_@@_testphase_table_bool
1692 { \UseTaggingSocket { tbl / hmode / begin } }
1693 \skip_horizontal:N \l_@@_left_margin_dim
1694 \skip_horizontal:N \l_@@_extra_left_margin_dim
1695 \c_math_toggle_token
1696 \bool_if:NTF \l_@@_light_syntax_bool
1697 { \use:c { @@-light-syntax } }
1698 { \use:c { @@-normal-syntax } }
1699 }
```

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

```
1700 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1701 {
1702 \tl_set:Nn \l_tmpa_tl { #1 }
1703 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1704 { \@@_rescan_for_spanish:N \l_tmpa_tl }
1705 \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1706 \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...

```
1707   \@@_pre_array:
1708   }
```

## 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).

```
1709 \cs_new_protected:Npn \@@_pre_code_before:
1710 {
```

First, we give values to the LaTeX counters `iRow` and `jCol`. We remind that, in the `\CodeBefore` (and in 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).

```
1711   \int_set:Nn \c{iRow} { \seq_item:Nn \g_@@_size_seq 2 }
1712   \int_set:Nn \c{jCol} { \seq_item:Nn \g_@@_size_seq 5 }
1713   \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1714   \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq 6 }
```

Now, we will create all the `col` nodes and `row` nodes with the informations written in the `aux` file. You use the technique described in the page 1229 of `pgfmanual.pdf`, version 3.1.4b.

```
1715   \pgfsys@markposition { \@@_env: - position }
1716   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1717   \pgfpicture
1718   \pgf@relevantforpicturesizefalse
```

First, the recreation of the `row` nodes.

```
1719   \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1720   {
1721     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1722     \pgfcoordinate { \@@_env: - row - ##1 }
1723     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1724   }
```

Now, the recreation of the `col` nodes.

```
1725   \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1726   {
1727     \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1728     \pgfcoordinate { \@@_env: - col - ##1 }
1729     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1730   }
```

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

```
1731   \@@_create_diag_nodes:
```

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

```
1732   \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1733   \endpgfpicture
```

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

```
1734   \@@_create_blocks_nodes:
```

```

1735 \IfPackageLoadedT { tikz }
1736 {
1737   \tikzset
1738   {
1739     every-picture / .style =
1740     { overlay , name-prefix = \@@_env: - }
1741   }
1742 }
1743 \cs_set_eq:NN \cellcolor \@@_cellcolor
1744 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1745 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1746 \cs_set_eq:NN \rowcolor \@@_rowcolor
1747 \cs_set_eq:NN \rowcolors \@@_rowcolors
1748 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1749 \cs_set_eq:NN \arraycolor \@@_arraycolor
1750 \cs_set_eq:NN \columncolor \@@_columncolor
1751 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1752 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1753 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1754 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1755 }

```

```

1756 \cs_new_protected:Npn \@@_exec_code_before:
1757 {

```

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...

```

1758 \clist_map_inline:Nn \l_@@_corners_cells_clist
1759 { \cs_set_nopar:cpn { @@_corner_ ##1 } { } }
1760 \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`.

```

1761 \@@_add_to_colors_seq:nn { { nocolor } } { }
1762 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1763 \group_begin:

```

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

```

1764 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token

```

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`).

```

1765 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1766 { \@@_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`.

```

1767 \exp_last_unbraced:No \@@_CodeBefore_keys:
1768 \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.

```

1769 \@@_actually_color:
1770 \l_@@_code_before_tl
1771 \q_stop

```

```

1772 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1773 \group_end:
1774 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1775 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1776 }

```

```

1777 \keys_define:nn { nicematrix / CodeBefore }
1778 {
1779   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1780   create-cell-nodes .default:n = true ,
1781   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1782   sub-matrix .value_required:n = true ,
1783   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1784   delimiters / color .value_required:n = true ,
1785   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1786 }
1787 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1788 {
1789   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1790   \@@_CodeBefore:w
1791 }

```

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.

```

1792 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1793 {
1794   \bool_if:NT \g_@@_aux_found_bool
1795   {
1796     \@@_pre_code_before:
1797     #1
1798   }
1799 }

```

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.

```

1800 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1801 {
1802   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1803   {
1804     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1805     \pgfcoordinate { \@@_env: - row - ##1 - base }
1806     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1807     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1808     {
1809       \cs_if_exist:cT
1810       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1811       {
1812         \pgfsys@getposition
1813         { \@@_env: - ##1 - #####1 - NW }
1814         \@@_node_position:
1815         \pgfsys@getposition
1816         { \@@_env: - ##1 - #####1 - SE }
1817         \@@_node_position_i:
1818         \@@_pgf_rect_node:nnn
1819         { \@@_env: - ##1 - #####1 }
1820         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1821         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1822       }
1823     }
1824 }

```

```

1824     }
1825 \int_step_inline:nn \c@iRow
1826 {
1827     \pgfnodealias
1828     { \@@_env: - ##1 - last }
1829     { \@@_env: - ##1 - \int_use:N \c@jCol }
1830 }
1831 \int_step_inline:nn \c@jCol
1832 {
1833     \pgfnodealias
1834     { \@@_env: - last - ##1 }
1835     { \@@_env: - \int_use:N \c@iRow - ##1 }
1836 }
1837 \@@_create_extra_nodes:
1838 }

```

```

1839 \cs_new_protected:Npn \@@_create_blocks_nodes:
1840 {
1841     \pgfpicture
1842     \pgf@relevantforpicturesizefalse
1843     \pgfrememberpicturepositiononpagetrue
1844     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1845     { \@@_create_one_block_node:nnnnn ##1 }
1846     \endpgfpicture
1847 }

```

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.<sup>6</sup>

```

1848 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1849 {
1850     \tl_if_empty:nF { #5 }
1851     {
1852         \@@_qpoint:n { col - #2 }
1853         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1854         \@@_qpoint:n { #1 }
1855         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1856         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1857         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1858         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1859         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1860         \@@_pgf_rect_node:nnnnn
1861         { \@@_env: - #5 }
1862         { \dim_use:N \l_tmpa_dim }
1863         { \dim_use:N \l_tmpb_dim }
1864         { \dim_use:N \l_@@_tmpc_dim }
1865         { \dim_use:N \l_@@_tmpd_dim }
1866     }
1867 }

```

```

1868 \cs_new_protected:Npn \@@_patch_for_revtext:
1869 {
1870     \cs_set_eq:NN \@addamp \@addamp@LaTeX
1871     \cs_set_eq:NN \insert@column \insert@column@array
1872     \cs_set_eq:NN \@classx \@classx@array
1873     \cs_set_eq:NN \@xarraycr \@xarraycr@array
1874     \cs_set_eq:NN \@arraycr \@arraycr@array
1875     \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1876     \cs_set_eq:NN \array \array@array

```

---

<sup>6</sup>Moreover, 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).

```

1877 \cs_set_eq:NN \@array \@array@array
1878 \cs_set_eq:NN \@tabular \@tabular@array
1879 \cs_set_eq:NN \@mkpream \@mkpream@array
1880 \cs_set_eq:NN \endarray \endarray@array
1881 \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1882 \cs_set:Npn \endtabular { \endarray $\egroup} % $
1883 }

```

## 10 The environment `{NiceArrayWithDelims}`

```

1884 \NewDocumentEnvironment { NiceArrayWithDelims }
1885 { m m 0 { } m ! 0 { } t \CodeBefore }
1886 {
1887 \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revteX:
1888 \@@_provide_pgfsyspdfmark:
1889 \bool_if:NT \g_@@_footnote_bool \savenotes

```

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

```

1890 \bgroup

1891 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1892 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1893 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1894 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

1895 \int_gzero:N \g_@@_block_box_int
1896 \dim_zero:N \g_@@_width_last_col_dim
1897 \dim_zero:N \g_@@_width_first_col_dim
1898 \bool_gset_false:N \g_@@_row_of_col_done_bool
1899 \str_if_empty:NT \g_@@_name_env_str
1900 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1901 \bool_if:NTF \l_@@_tabular_bool
1902 \mode_leave_vertical:
1903 \@@_test_if_math_mode:
1904 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet-in-env } }
1905 \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array<sup>7</sup>. 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.

```

1906 \cs_gset_eq:NN \@@_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.

```

1907 \cs_if_exist:NT \tikz@library@external@loaded
1908 {
1909 \tikzexternaldisable
1910 \cs_if_exist:NT \ifstandalone
1911 { \tikzset { external / optimize = false } }
1912 }

```

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

---

<sup>7</sup>e.g. `\color[rgb]{0.5,0.5,0}`

```

1913 \int_gincr:N \g_@@_env_int
1914 \bool_if:NF \l_@@_block_auto_columns_width_bool
1915 { \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 (except the blocks with the key `hvlines`).

```

1916 \seq_gclear:N \g_@@_blocks_seq
1917 \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`.

```

1918 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1919 \seq_gclear:N \g_@@_pos_of_xdots_seq
1920 \tl_gclear_new:N \g_@@_code_before_tl
1921 \tl_gclear:N \g_@@_row_style_tl

```

We load all the informations written in the aux file during previous compilations corresponding to the current environment.

```

1922 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1923 {
1924   \bool_gset_true:N \g_@@_aux_found_bool
1925   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1926 }
1927 { \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.

```

1928 \tl_gclear:N \g_@@_aux_tl
1929 \tl_if_empty:NF \g_@@_code_before_tl
1930 {
1931   \bool_set_true:N \l_@@_code_before_bool
1932   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1933 }
1934 \tl_if_empty:NF \g_@@_pre_code_before_tl
1935 { \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`.

```

1936 \bool_if:NTF \g_@@_delims_bool
1937 { \keys_set:nn { nicematrix / pNiceArray } }
1938 { \keys_set:nn { nicematrix / NiceArray } }
1939 { #3 , #5 }

```

```

1940 \@@_set_CT@arc@:o \l_@@_rules_color_tl

```

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:`.

```

1941 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1942 }

```

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

```

1943 {
1944   \bool_if:NTF \l_@@_light_syntax_bool
1945   { \use:c { end @@-light-syntax } }
1946   { \use:c { end @@-normal-syntax } }
1947   \c_math_toggle_token
1948   \skip_horizontal:N \l_@@_right_margin_dim
1949   \skip_horizontal:N \l_@@_extra_right_margin_dim

```

```

1950
1951 % awful workaround
1952 \int_compare:nNnT \g_@@_col_total_int = \c_one_int
1953 {
1954   \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim
1955   {
1956     \skip_horizontal:N - \l_@@_columns_width_dim
1957     \bool_if:NTF \l_@@_tabular_bool
1958       { \skip_horizontal:n { - 2 \tabcolsep } }
1959       { \skip_horizontal:n { - 2 \arraycolsep } }
1960   }
1961 }
1962 \hbox_set_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.

```

1963 \bool_if:NT \l_@@_width_used_bool
1964 {
1965   \int_if_zero:nT \g_@@_total_X_weight_int
1966   { \@@_error_or_warning:n { width-without-X-columns } }
1967 }

```

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. For a `X`-column of weight  $n$ , the width will be `\l_@@_X_columns_dim` multiplied by  $n$ .

```

1968 \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
1969 {
1970   \tl_gput_right:Ne \g_@@_aux_tl
1971   {
1972     \bool_set_true:N \l_@@_X_columns_aux_bool
1973     \dim_set:Nn \l_@@_X_columns_dim
1974     {
1975       \dim_compare:nNnTF
1976         {
1977           \dim_abs:n
1978             { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1979         }
1980         <
1981         { 0.001 pt }
1982         { \dim_use:N \l_@@_X_columns_dim }
1983         {
1984           \dim_eval:n
1985             {
1986               ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1987               / \int_use:N \g_@@_total_X_weight_int
1988               + \l_@@_X_columns_dim
1989             }
1990         }
1991     }
1992   }
1993 }

```

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).

```

1994 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1995 {
1996   \bool_if:NF \l_@@_last_row_without_value_bool
1997   {
1998     \int_compare:nNnF \l_@@_last_row_int = \c_iRow
1999     {
2000       \@@_error:n { Wrong-last-row }
2001       \int_gset_eq:NN \l_@@_last_row_int \c_iRow

```

```

2002     }
2003   }
2004 }

```

Now, the definition of `\c@jCol` and `\g_@@_col_total_int` change: `\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”.<sup>8</sup>

```

2005   \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2006   \bool_if:NTF \g_@@_last_col_found_bool
2007     { \int_gdecr:N \c@jCol }
2008   {
2009     \int_compare:nNnT \l_@@_last_col_int > { -1 }
2010     { \@@_error:n { last~col~not~used } }
2011   }

```

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

```

2012   \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2013   \int_compare:nNnT \l_@@_last_row_int > { -1 } { \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. 89).

```

2014   \int_if_zero:nT \l_@@_first_col_int
2015     { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2016   \bool_if:nTF { ! \g_@@_delims_bool }
2017   {
2018     \str_if_eq:eeTF \l_@@_baseline_tl { c }
2019     \@@_use_arraybox_with_notes_c:
2020     {
2021       \str_if_eq:eeTF \l_@@_baseline_tl { b }
2022       \@@_use_arraybox_with_notes_b:
2023       \@@_use_arraybox_with_notes:
2024     }
2025   }

```

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).

```

2026   {
2027     \int_if_zero:nTF \l_@@_first_row_int
2028     {
2029       \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2030       \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2031     }
2032     { \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”.<sup>9</sup>

```

2033     \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2034     {
2035       \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2036       \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2037     }
2038     { \dim_zero:N \l_tmpb_dim }
2039   \hbox_set:Nn \l_tmpa_box
2040   {
2041     \c_math_toggle_token
2042     \@@_color:o \l_@@_delimiters_color_tl
2043     \exp_after:wN \left \g_@@_left_delim_tl
2044     \vcenter

```

<sup>8</sup>We remind that the potential “first column” (exterior) has the number 0.

<sup>9</sup>A 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).



2045

{

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.

2046

`\skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }`

2047

`\hbox`

2048

{

2049

`\bool_if:NTF \l_@@_tabular_bool`

2050

`{ \skip_horizontal:N -\tabcolsep }`

2051

`{ \skip_horizontal:N -\arraycolsep }`

2052

`\@@_use_arraybox_with_notes_c:`

2053

`\bool_if:NTF \l_@@_tabular_bool`

2054

`{ \skip_horizontal:N -\tabcolsep }`

2055

`{ \skip_horizontal:N -\arraycolsep }`

2056

}

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

2057

`\skip_vertical:N -\l_tmpb_dim`

2058

`\skip_vertical:N \arrayrulewidth`

2059

}

2060

`\exp_after:wN \right \g_@@_right_delim_tl`

2061

`\c_math_toggle_token`

2062

}

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.

2063

`\bool_if:NTF \l_@@_delimiters_max_width_bool`

2064

{

2065

`\@@_put_box_in_flow_bis:nn`

2066

`\g_@@_left_delim_tl`

2067

`\g_@@_right_delim_tl`

2068

}

2069

`\@@_put_box_in_flow:`

2070

}

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. 90).

2071

`\bool_if:NT \g_@@_last_col_found_bool`

2072

`{ \skip_horizontal:N \g_@@_width_last_col_dim }`

2073

`\bool_if:NT \l_@@_preamble_bool`

2074

{

2075

`\int_compare:nNnT \c_jCol < \g_@@_static_num_of_col_int`

2076

`{ \@@_warning_gredirect_none:n { columns-not-used } }`

2077

}

2078

`\@@_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 exponent to a matrix in a mathematical formula.

2079

`\egroup`

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

2080

`\iow_now:Nn \@mainaux { \ExplSyntaxOn }`

2081

`\iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }`

2082

`\iow_now:Ne \@mainaux`

2083

{

2084

`\tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }`

2085

`{ \exp_not:o \g_@@_aux_tl }`

2086

}

2087

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

2088

`\bool_if:NT \g_@@_footnote_bool \endsavenotes`

2089

}

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

## 11 We construct 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` also.

```
2090 \cs_new_protected:Npn \@@_transform_preamble:
2091   {
2092     \@@_transform_preamble_i:
2093     \@@_transform_preamble_ii:
2094   }
2095 \cs_new_protected:Npn \@@_transform_preamble_i:
2096   {
2097     \int_gzero:N \c@jCol
```

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

```
2098   \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.

```
2099   \bool_gset_false:N \g_tmpb_bool
```

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

```
2100   \tl_gclear_new:N \g_@@_pre_cell_tl
```

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

```
2101   \int_zero:N \l_tmpa_int
2102   \tl_gclear:N \g_@@_array_preamble_tl
2103   \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2104     {
2105       \tl_gset:Nn \g_@@_array_preamble_tl
2106         { ! { \skip_horizontal:N \arrayrulewidth } }
2107     }
2108     {
2109       \clist_if_in:NnT \l_@@_vlines_clist 1
2110       {
2111         \tl_gset:Nn \g_@@_array_preamble_tl
2112           { ! { \skip_horizontal:N \arrayrulewidth } }
2113       }
2114     }
```

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

```
2115   \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \@@_stop:
2116   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
2117   \@@_replace_columncolor:
2118 }
```

```
2119 \hook_gput_code:nnn { begindocument } { . }
2120 {
2121   \IfPackageLoadedTF { colortbl }
2122   {
```

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).

```

2123     \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2124     \cs_new_protected:Npn \@@_replace_columncolor:
2125     {
2126         \regex_replace_all:NnN
2127         \c_@@_columncolor_regex
2128         { \c { @@_columncolor_preamble } }
2129         \g_@@_array_preamble_tl
2130     }
2131 }
2132 {
2133     \cs_new_protected:Npn \@@_replace_columncolor:
2134     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2135 }
2136 }

```

```

2137 \cs_new_protected:Npn \@@_transform_preamble_ii:
2138 {

```

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

```

2139     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2140     {
2141         \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2142         { \bool_gset_true:N \g_@@_delims_bool }
2143     }
2144     { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2145     \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).

```

2146     \int_if_zero:nTF \l_@@_first_col_int
2147     { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2148     {
2149         \bool_if:NF \g_@@_delims_bool
2150         {
2151             \bool_if:NF \l_@@_tabular_bool
2152             {
2153                 \clist_if_empty:NT \l_@@_vlines_clist
2154                 {
2155                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2156                     { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2157                 }
2158             }
2159         }
2160     }
2161     \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2162     { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2163     {
2164         \bool_if:NF \g_@@_delims_bool
2165         {
2166             \bool_if:NF \l_@@_tabular_bool
2167             {
2168                 \clist_if_empty:NT \l_@@_vlines_clist
2169                 {
2170                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2171                     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2172                 }
2173             }
2174         }

```

```

2173     }
2174   }
2175 }

```

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`).

```

2176   \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2177   {
2178     \tl_gput_right:Nn \g_@@_array_preamble_tl
2179     { > { \@@_error_too_much_cols: } 1 }
2180   }
2181 }

```

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.

```

2182 \cs_new_protected:Npn \@@_rec_preamble:n #1
2183 {

```

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.<sup>10</sup>

```

2184   \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2185   { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2186   {

```

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

```

2187     \cs_if_exist:cTF { NC @ find @ #1 }
2188     {
2189       \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2190       \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2191     }
2192     {
2193       \str_if_eq:nnTF { #1 } { S }
2194       { \@@_fatal:n { unknown~column~type~S } }
2195       { \@@_fatal:nn { unknown~column~type } { #1 } }
2196     }
2197   }
2198 }

```

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

```

2199 \cs_new:Npn \@@_c #1
2200 {
2201   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2202   \tl_gclear:N \g_@@_pre_cell_tl
2203   \tl_gput_right:Nn \g_@@_array_preamble_tl
2204   { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2205   \int_gincr:N \c@jCol
2206   \@@_rec_preamble_after_col:n
2207 }

2208 \cs_new:Npn \@@_l #1
2209 {
2210   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2211   \tl_gclear:N \g_@@_pre_cell_tl
2212   \tl_gput_right:Nn \g_@@_array_preamble_tl

```

---

<sup>10</sup>We 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`.

```

2213     {
2214     > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2215     l
2216     < \@@_cell_end:
2217     }
2218     \int_gincr:N \c@jCol
2219     \@@_rec_preamble_after_col:n
2220   }
2221 \cs_new:Npn \@@_r #1
2222 {
2223   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2224   \tl_gclear:N \g_@@_pre_cell_tl
2225   \tl_gput_right:Nn \g_@@_array_preamble_tl
2226   {
2227     > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2228     r
2229     < \@@_cell_end:
2230   }
2231   \int_gincr:N \c@jCol
2232   \@@_rec_preamble_after_col:n
2233 }

```

For ! and @

```

2234 \cs_new:cpn { @@ _ \token_to_str:N ! } #1 #2
2235 {
2236   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2237   \@@_rec_preamble:n
2238 }
2239 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2240 \cs_new:cpn { @@ _ | } #1
2241 {

```

\l\_tmpa\_int is the number of successive occurrences of |

```

2242   \int_incr:N \l_tmpa_int
2243   \@@_make_preamble_i_i:n
2244 }
2245 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2246 {
2247   \str_if_eq:nnTF { #1 } { | }
2248   { \use:c { @@ _ | } | }
2249   { \@@_make_preamble_i_ii:nn { } #1 }
2250 }
2251 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2252 {
2253   \str_if_eq:nnTF { #2 } { [ ]
2254   { \@@_make_preamble_i_ii:nw { #1 } [ ]
2255   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2256 }
2257 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2258 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2259 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2260 {
2261   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2262   \tl_gput_right:Ne \g_@@_array_preamble_tl
2263   {

```

Here, the command \dim\_use:N is mandatory.

```

2264     \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2265     }
2266   \tl_gput_right:Ne \g_@@_pre_code_after_tl

```

```

2267 {
2268   \@@_vline:n
2269   {
2270     position = \int_eval:n { \c@jCol + 1 } ,
2271     multiplicity = \int_use:N \l_tmpa_int ,
2272     total-width = \dim_use:N \l_@@_rule_width_dim ,
2273     #2
2274   }

```

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.

```

2275 }
2276 \int_zero:N \l_tmpa_int
2277 \str_if_eq:nnT { #1 } { \@@_stop: } { \bool_gset_true:N \g_tmpb_bool }
2278 \@@_rec_preamble:n #1
2279 }

```

```

2280 \cs_new:cpn { @@_ > } #1 #2
2281 {
2282   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2283   \@@_rec_preamble:n
2284 }

```

```

2285 \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.

```

2286 \keys_define:nn { nicematrix / p-column }
2287 {
2288   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2289   r .value_forbidden:n = true ,
2290   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2291   c .value_forbidden:n = true ,
2292   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2293   l .value_forbidden:n = true ,
2294   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2295   S .value_forbidden:n = true ,
2296   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2297   p .value_forbidden:n = true ,
2298   t .meta:n = p ,
2299   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2300   m .value_forbidden:n = true ,
2301   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2302   b .value_forbidden:n = true
2303 }

```

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

```

2304 \cs_new:Npn \@@_p #1
2305 {
2306   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2307   \@@_make_preamble_ii_i:n
2308 }
2309 \cs_set_eq:NN \@@_b \@@_p
2310 \cs_set_eq:NN \@@_m \@@_p
2311 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2312 {
2313   \str_if_eq:nnTF { #1 } { [ ]
2314     { \@@_make_preamble_ii_ii:w [ ]
2315       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2316   }

```

```

2317 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2318 { \@@_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.

```

2319 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2320 {

```

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).

```

2321   \str_set:Nn \l_@@_hpos_col_str { j }
2322   \@@_keys_p_column:n { #1 }
2323   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2324 }

2325 \cs_new_protected:Npn \@@_keys_p_column:n #1
2326 { \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.

```

2327 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2328 {
2329   \use:e
2330   {
2331     \@@_make_preamble_ii_v:nnnnnnn
2332     { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }
2333     { \dim_eval:n { #1 } }
2334     {

```

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.

```

2335         \str_if_eq:nnTF \l_@@_hpos_col_str { j }
2336         { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2337         {

```

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

```

2338             \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2339             { \str_lowercase:o \l_@@_hpos_col_str }
2340         }
2341     \IfPackageLoadedTF { ragged2e }
2342     {
2343       \str_case:on \l_@@_hpos_col_str
2344       {
2345         c { \exp_not:N \Centering }
2346         l { \exp_not:N \RaggedRight }
2347         r { \exp_not:N \RaggedLeft }
2348       }
2349     }
2350     {
2351       \str_case:on \l_@@_hpos_col_str
2352       {
2353         c { \exp_not:N \centering }
2354         l { \exp_not:N \raggedright }
2355         r { \exp_not:N \raggedleft }
2356       }
2357     }
2358     #3
2359   }
2360   { \str_if_eq:nnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2361   { \str_if_eq:nnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2362   { \str_if_eq:nnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2363   { #2 }

```

```

2364     {
2365     \str_case:onF \l_@@_hpos_col_str
2366     {
2367     { j } { c }
2368     { si } { c }
2369     }

```

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

```

2370     { \str_lowercase:o \l_@@_hpos_col_str }
2371     }
2372 }

```

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

```

2373 \int_gincr:N \c_jCol
2374 \@@_rec_preamble_after_col:n
2375 }

```

#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*.

```

2376 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2377 {
2378 \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2379 {
2380 \tl_gput_right:Nn \g_@@_array_preamble_tl
2381 { > \@@_test_if_empty_for_S: }
2382 }
2383 {
2384 \tl_gput_right:Nn \g_@@_array_preamble_tl
2385 { > \@@_test_if_empty: }
2386 }
2387 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2388 \tl_gclear:N \g_@@_pre_cell_tl
2389 \tl_gput_right:Nn \g_@@_array_preamble_tl
2390 {
2391 > {

```

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.

```

2392 \dim_set:Nn \l_@@_col_width_dim { #2 }
2393 \bool_if:NT \c_@@_testphase_table_bool
2394 { \tag_struct_begin:n { tag = Div } }
2395 \@@_cell_begin:

```

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

```

2396 \use:c { #7 } [ #1 ] { #2 }

```

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

```

2397 \everypar
2398 {
2399 \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2400 \everypar { }
2401 }
2402 \bool_if:NT \c_@@_testphase_table_bool \tagpdfpara0n

```



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

```
2403         #3
```

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

```
2404         \g_@@_row_style_tl
2405         \arraybackslash
2406         #5
2407     }
2408     #8
2409     < {
2410         #6
```

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

```
2411         \@finalstrut \@arstrutbox
2412         \use:c { end #7 }
```

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

```
2413         #4
2414         \@@_cell_end:
2415         \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2416     }
2417 }
2418 }
```

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

```
2419 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2420 {
```

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 `&`).

```
2421     \group_align_safe_begin:
2422     \peek_meaning:NTF &
2423     {
2424         \group_align_safe_end:
2425         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2426         {
```

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

```
2427         \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2428         \skip_horizontal:N \l_@@_col_width_dim
2429     }
2430 }
2431 { \group_align_safe_end: }
2432 }

2433 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2434 {
2435     \peek_meaning:NT \__siunitx_table_skip:n
2436     { \bool_gset_true:N \g_@@_empty_cell_bool }
2437 }
```

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

```
2438 \cs_new_protected:Npn \@@_center_cell_box:
2439 {
```

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

```

2440 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2441 {
2442   \int_compare:nNnT
2443     { \box_ht:N \l_@@_cell_box }
2444     >

```

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).

```

2445     { \box_ht:N \strutbox }
2446     {
2447       \hbox_set:Nn \l_@@_cell_box
2448         {
2449           \box_move_down:nn
2450             {
2451               ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2452                 + \baselineskip ) / 2
2453             }
2454             { \box_use:N \l_@@_cell_box }
2455           }
2456         }
2457     }
2458 }

```

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

```

2459 \cs_new:Npn \@@_V #1 #2
2460 {
2461   \str_if_eq:nnTF { #1 } { [ ]
2462     { \@@_make_preamble_V_i:w [ ]
2463       { \@@_make_preamble_V_i:w [ ] { #2 } }
2464     }
2465   \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2466     { \@@_make_preamble_V_ii:nn { #1 } }
2467   \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2468     {
2469       \str_set:Nn \l_@@_vpos_col_str { p }
2470       \str_set:Nn \l_@@_hpos_col_str { j }
2471       \@@_keys_p_column:n { #1 }
2472       \IfPackageLoadedTF { varwidth }
2473         { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2474         {
2475           \@@_error_or_warning:n { varwidth-not-loaded }
2476           \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2477         }
2478     }

```

For w and W

```

2479 \cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2480 \cs_new: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.

```

2481 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2482 {
2483   \str_if_eq:nnTF { #3 } { s }
2484     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2485     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2486 }

```

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.

```

2487 \cs_new_protected:Npn \@@_make_preamble_w_i:nmmm #1 #2
2488 {
2489   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2490   \tl_gclear:N \g_@@_pre_cell_tl
2491   \tl_gput_right:Nn \g_@@_array_preamble_tl
2492     {
2493       > {
2494         \dim_set:Nn \l_@@_col_width_dim { #2 }
2495         \@@_cell_begin:
2496         \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2497       }
2498       c
2499       < {
2500         \@@_cell_end_for_w_s:
2501         #1
2502         \@@_adjust_size_box:
2503         \box_use_drop:N \l_@@_cell_box
2504       }
2505     }
2506   \int_gincr:N \c@jCol
2507   \@@_rec_preamble_after_col:n
2508 }

```

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

```

2509 \cs_new_protected:Npn \@@_make_preamble_w_ii:nmmm #1 #2 #3 #4
2510 {
2511   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2512   \tl_gclear:N \g_@@_pre_cell_tl
2513   \tl_gput_right:Nn \g_@@_array_preamble_tl
2514     {
2515       > {

```

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.

```

2516         \dim_set:Nn \l_@@_col_width_dim { #4 }
2517         \hbox_set:Nw \l_@@_cell_box
2518         \@@_cell_begin:
2519         \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2520       }
2521       c
2522       < {
2523         \@@_cell_end:
2524         \hbox_set_end:
2525         #1
2526         \@@_adjust_size_box:
2527         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2528       }
2529     }

```

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

```

2530   \int_gincr:N \c@jCol
2531   \@@_rec_preamble_after_col:n
2532 }

```

```

2533 \cs_new_protected:Npn \@@_special_W:
2534 {
2535   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2536     { \@@_warning:n { W~warning } }
2537 }

```

For S (of siunitx).

```

2538 \cs_new:Npn \@@_S #1 #2
2539 {
2540   \str_if_eq:nnTF { #1 } { [ ]
2541     { \@@_make_preamble_S:w [ ]
2542       { \@@_make_preamble_S:w [ ] { #2 } }
2543     }
2544   \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2545     { \@@_make_preamble_S_i:n { #1 } }
2546   \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2547     {
2548       \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2549       \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2550       \tl_gc_clear:N \g_@@_pre_cell_tl
2551       \tl_gput_right:Nn \g_@@_array_preamble_tl
2552         {
2553           > {
2554             \@@_cell_begin:
2555             \keys_set:nn { siunitx } { #1 }
2556             \siunitx_cell_begin:w
2557           }
2558           c
2559           < { \siunitx_cell_end: \@@_cell_end: }
2560         }

```

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

```

2561   \int_gincr:N \c@jCol
2562   \@@_rec_preamble_after_col:n
2563 }

```

For (, [ and \{.

```

2564 \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2565 {
2566   \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.

```

2567   \int_if_zero:nTF \c@jCol
2568   {
2569     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2570     {

```

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

```

2571       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2572       \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2573       \@@_rec_preamble:n #2
2574     }
2575     {
2576       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2577       \@@_make_preamble_iv:nn { #1 } { #2 }
2578     }
2579   }
2580   { \@@_make_preamble_iv:nn { #1 } { #2 } }
2581 }
2582 \cs_set_eq:cc { @@ _ \token_to_str:N [ ] { @@ _ \token_to_str:N ( }
2583 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2584 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2585 {
2586   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2587     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2588   \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2589   {
2590     \@@_error:nn { delimiter~after~opening } { #2 }

```

```

2591     \@@_rec_preamble:n
2592   }
2593   { \@@_rec_preamble:n #2 }
2594 }

```

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

```

2595 \cs_new:cpn { @@ _ \token_to_str:N \left } #1 { \use:c { @@ _ \token_to_str:N ( } }

```

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

```

2596 \cs_new:cpn { @@ _ \token_to_str:N ) } #1 #2
2597   {
2598     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2599     \tl_if_in:nnTF { ) ] \ } { #2 }
2600     { \@@_make_preamble_v:nnn #1 #2 }
2601     {
2602       \str_if_eq:nnTF { \@@_stop: } { #2 }
2603       {
2604         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2605         { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2606         {
2607           \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2608           \tl_gput_right:Ne \g_@@_pre_code_after_tl
2609             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2610           \@@_rec_preamble:n #2
2611         }
2612       }
2613       {
2614         \tl_if_in:nnT { ( [ \{ \left } { #2 }
2615         { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2616         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2617           { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2618         \@@_rec_preamble:n #2
2619       }
2620     }
2621   }
2622 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ) }
2623 \cs_set_eq:cc { @@ _ \token_to_str:N \ } { @@ _ \token_to_str:N ) }
2624 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2625   {
2626     \str_if_eq:nnTF { \@@_stop: } { #3 }
2627     {
2628       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2629       {
2630         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2631         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2632           { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2633         \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2634       }
2635       {
2636         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2637         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2638           { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2639         \@@_error:nn { double~closing~delimiter } { #2 }
2640       }
2641     }
2642     {
2643       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2644         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2645       \@@_error:nn { double~closing~delimiter } { #2 }
2646       \@@_rec_preamble:n #3

```

```

2647     }
2648   }

2649 \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2650 { \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 @{...}.

```

2651 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2652 {
2653   \str_if_eq:nnTF { #1 } { < }
2654   \@@_rec_preamble_after_col_i:n
2655   {
2656     \str_if_eq:nnTF { #1 } { @ }
2657     \@@_rec_preamble_after_col_ii:n
2658     {
2659       \str_if_eq:nnTF \l_@@_vlines_clist { all }
2660       {
2661         \tl_gput_right:Nn \g_@@_array_preamble_tl
2662         { ! { \skip_horizontal:N \arrayrulewidth } }
2663       }
2664       {
2665         \clist_if_in:NeT \l_@@_vlines_clist
2666         { \int_eval:n { \c@jCol + 1 } }
2667         {
2668           \tl_gput_right:Nn \g_@@_array_preamble_tl
2669           { ! { \skip_horizontal:N \arrayrulewidth } }
2670         }
2671       }
2672       \@@_rec_preamble:n { #1 }
2673     }
2674   }
2675 }

2676 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2677 {
2678   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2679   \@@_rec_preamble_after_col:n
2680 }

```

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.

```

2681 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2682 {
2683   \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2684   {
2685     \tl_gput_right:Nn \g_@@_array_preamble_tl
2686     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2687   }
2688   {
2689     \clist_if_in:NeTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2690     {
2691       \tl_gput_right:Nn \g_@@_array_preamble_tl
2692       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2693     }
2694     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2695   }
2696   \@@_rec_preamble:n
2697 }

2698 \cs_new:cpn { @@ _ * } #1 #2 #3

```

```

2699 {
2700   \tl_clear:N \l_tmpa_tl
2701   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2702   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2703 }

```

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

```

2704 \cs_new:cpn { @@ _ \token_to_str:N \NC@find } #1 { \@@_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.

```

2705 \cs_new:Npn \@@_X #1 #2
2706 {
2707   \str_if_eq:nnTF { #2 } { [ ]
2708     { \@@_make_preamble_X:w [ ] }
2709     { \@@_make_preamble_X:w [ ] #2 }
2710 }
2711 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2712 { \@@_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 as 1, 2, 3, etc. The following set of keys will be used to retrieve that value (in the counter `\l_@@_weight_int`).

```

2713 \keys_define:nn { nicematrix / X-column }
2714 { unknown .code:n = \int_set:Nn \l_@@_weight_int { \l_keys_key_str } }

```

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

```

2715 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2716 {

```

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).

```

2717   \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).

```

2718   \str_set:Nn \l_@@_vpos_col_str { p }

```

The integer `\l_@@_weight_int` will be the weight of the X column (the initial value is 1). The user may specify a different value (such as 2, 3, etc.) by putting that value in the optional argument of the specifier. The weights of the X columns are used in the computation of the actual width of those columns as in `tabu` (now obsolete) or `tabularray`.

```

2719   \int_zero_new:N \l_@@_weight_int
2720   \int_set_eq:NN \l_@@_weight_int \c_one_int
2721   \@@_keys_p_column:n { #1 }

```

The unknown keys are put in `\l_tmpa_tl`

```

2722   \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2723   \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2724     {
2725       \@@_error_or_warning:n { negative-weight }
2726       \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2727     }
2728   \int_gadd:Nn \g_@@_total_X_weight_int \l_@@_weight_int

```

We test whether we know the 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).

```

2729   \bool_if:NTF \l_@@_X_columns_aux_bool
2730   {
2731     \@@_make_preamble_ii_iv:nnn
2732     { \l_@@_weight_int \l_@@_X_columns_dim }
2733     { minipage }
2734     { \@@_no_update_width: }
2735   }
2736   {
2737     \tl_gput_right:Nn \g_@@_array_preamble_tl
2738     {
2739       > {
2740         \@@_cell_begin:
2741         \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`.

```

2742   \NotEmpty

```

The following code will nullify the box of the cell.

```

2743     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2744     { \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`.

```

2745     \begin { minipage } { 5 cm } \arraybackslash
2746     }
2747     c
2748     < {
2749     \end { minipage }
2750     \@@_cell_end:
2751     }
2752   }
2753   \int_gincr:N \c@jCol
2754   \@@_rec_preamble_after_col:n
2755 }
2756 }

```

```

2757 \cs_new_protected:Npn \@@_no_update_width:
2758 {
2759   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2760   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2761 }

```

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

```

2762 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2763 {
2764   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2765   { \int_eval:n { \c@jCol + 1 } }
2766   \tl_gput_right:Ne \g_@@_array_preamble_tl
2767   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2768   \@@_rec_preamble:n
2769 }

```

The token `\@@_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.

```

2770 \cs_set_eq:cN { @@ _ \token_to_str:N \@@_stop: } \use_none:n

```



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

```

2771 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2772   { \@@_fatal:n { Preamble-forgotten } }
2773 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2774 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2775 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore } { @@ _ \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).

```

2776 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2777   {

```

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`.

```

2778   \multispan { #1 }
2779   \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2780   \begingroup
2781   \bool_if:NT \c_@@_testphase_table_bool
2782     { \tbl_update_multicolumn_cell_data:n { #1 } }
2783   \cs_set_nopar:Npn \@addamp
2784     { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2785   \tl_gclear:N \g_@@_preamble_tl
2786   \@@_make_m_preamble:n #2 \q_stop

```

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

```

2787   \exp_args:No \@mkpream \g_@@_preamble_tl
2788   \@addtopreamble \@empty
2789   \endgroup
2790   \bool_if:NT \c_@@_testphase_table_bool
2791     { \UseTaggingSocket { tbl / colspan } { #1 } }

```

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

```

2792   \int_compare:nNnT { #1 } > \c_one_int
2793   {
2794     \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2795       { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2796     \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2797     \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2798       {
2799         {
2800           \int_if_zero:nTF \c@jCol
2801             { \int_eval:n { \c@iRow + 1 } }
2802             { \int_use:N \c@iRow }
2803         }
2804         { \int_eval:n { \c@jCol + 1 } }
2805         {
2806           \int_if_zero:nTF \c@jCol
2807             { \int_eval:n { \c@iRow + 1 } }
2808             { \int_use:N \c@iRow }
2809         }
2810         { \int_eval:n { \c@jCol + #1 } }
2811         { } % for the name of the block

```

```

2812     }
2813 }

```

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

```

2814 \RenewDocumentCommand \cellcolor { 0 { } m }
2815 {
2816   \@@_test_color_inside:
2817   \tl_gput_right:Ne \g_@@_pre_code_before_tl
2818   {
2819     \@@_rectanglecolor [ ##1 ]
2820     { \exp_not:n { ##2 } }
2821     { \int_use:N \c@iRow - \int_use:N \c@jCol }
2822     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2823   }
2824   \ignorespaces
2825 }

```

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

```

2826 \cs_set_nopar:Npn \@sharp { #3 }
2827 \@arstrut
2828 \@preamble
2829 \null

```

We add some lines.

```

2830 \int_gadd:Nn \c@jCol { #1 - 1 }
2831 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2832   { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2833 \ignorespaces
2834 }

```

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`.

```

2835 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2836 {
2837   \str_case:nnF { #1 }
2838   {
2839     c { \@@_make_m_preamble_i:n #1 }
2840     l { \@@_make_m_preamble_i:n #1 }
2841     r { \@@_make_m_preamble_i:n #1 }
2842     > { \@@_make_m_preamble_ii:nn #1 }
2843     ! { \@@_make_m_preamble_ii:nn #1 }
2844     @ { \@@_make_m_preamble_ii:nn #1 }
2845     | { \@@_make_m_preamble_iii:n #1 }
2846     p { \@@_make_m_preamble_iv:nnn t #1 }
2847     m { \@@_make_m_preamble_iv:nnn c #1 }
2848     b { \@@_make_m_preamble_iv:nnn b #1 }
2849     w { \@@_make_m_preamble_v:nnnn { } #1 }
2850     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2851     \q_stop { }
2852   }
2853   {
2854     \cs_if_exist:cTF { NC @ find @ #1 }
2855     {
2856       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2857       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2858     }
2859     {
2860       \str_if_eq:nnTF { #1 } { S }
2861       { \@@_fatal:n { unknown~column~type~S } }
2862       { \@@_fatal:nn { unknown~column~type } { #1 } }

```

```

2863     }
2864   }
2865 }

```

For c, l and r

```

2866 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2867 {
2868   \tl_gput_right:Nn \g_@@_preamble_tl
2869   {
2870     > { \@@_cell_begin: \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2871     #1
2872     < \@@_cell_end:
2873   }

```

We test for the presence of a <.

```

2874   \@@_make_m_preamble_x:n
2875 }

```

For >, ! and @

```

2876 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2877 {
2878   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2879   \@@_make_m_preamble:n
2880 }

```

For |

```

2881 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2882 {
2883   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2884   \@@_make_m_preamble:n
2885 }

```

For p, m and b

```

2886 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2887 {
2888   \tl_gput_right:Nn \g_@@_preamble_tl
2889   {
2890     > {
2891       \@@_cell_begin:
2892       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2893       \mode_leave_vertical:
2894       \arraybackslash
2895       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2896     }
2897     c
2898     < {
2899       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2900       \end { minipage }
2901       \@@_cell_end:
2902     }
2903   }

```

We test for the presence of a <.

```

2904   \@@_make_m_preamble_x:n
2905 }

```

For w and W

```

2906 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2907 {
2908   \tl_gput_right:Nn \g_@@_preamble_tl
2909   {
2910     > {
2911       \dim_set:Nn \l_@@_col_width_dim { #4 }
2912       \hbox_set:Nw \l_@@_cell_box

```

```

2913     \@@_cell_begin:
2914     \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2915   }
2916   c
2917   < {
2918     \@@_cell_end:
2919     \hbox_set_end:
2920     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2921     #1
2922     \@@_adjust_size_box:
2923     \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2924   }
2925 }

```

We test for the presence of a <.

```

2926   \@@_make_m_preamble_x:n
2927 }

```

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

```

2928 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2929 {
2930   \str_if_eq:nnTF { #1 } { < }
2931   \@@_make_m_preamble_ix:n
2932   { \@@_make_m_preamble:n { #1 } }
2933 }
2934 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2935 {
2936   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2937   \@@_make_m_preamble_x:n
2938 }

```

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`).

```

2939 \cs_new_protected:Npn \@@_put_box_in_flow:
2940 {
2941   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2942   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2943   \str_if_eq:eeTF \l_@@_baseline_tl { c }
2944   { \box_use_drop:N \l_tmpa_box }
2945   \@@_put_box_in_flow_i:
2946 }

```

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

```

2947 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2948 {
2949   \pgfpicture
2950     \@@_qpoint:n { row - 1 }
2951     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2952     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2953     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2954     \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).

```

2955   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
2956   {
2957     \int_set:Nn \l_tmpa_int
2958     {
2959       \str_range:Nnn

```

```

2960         \l_@@_baseline_tl
2961         6
2962         { \tl_count:o \l_@@_baseline_tl }
2963     }
2964     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2965 }
2966 {
2967     \str_if_eq:eeTF \l_@@_baseline_tl { t }
2968     { \int_set_eq:NN \l_tmpa_int \c_one_int }
2969     {
2970         \str_if_eq:onTF \l_@@_baseline_tl { b }
2971         { \int_set_eq:NN \l_tmpa_int \c@iRow }
2972         { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2973     }
2974     \bool_lazy_or:nnT
2975     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2976     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2977     {
2978         \@@_error:n { bad-value-for-baseline }
2979         \int_set_eq:NN \l_tmpa_int \c_one_int
2980     }
2981     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2982         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2983     }
2984     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2985     \endpgfpicture
2986     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2987     \box_use_drop:N \l_tmpa_box
2988 }

```

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).

```

2989 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2990 {

```

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.

```

2991     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
2992     {
2993         \int_compare:nNnT \c@jCol > \c_one_int
2994         {
2995             \box_set_wd:Nn \l_@@_the_array_box
2996             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2997         }
2998     }

```

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

```

2999     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3000     \bool_if:NT \l_@@_caption_above_bool
3001     {
3002         \tl_if_empty:NF \l_@@_caption_tl
3003         {
3004             \bool_set_false:N \g_@@_caption_finished_bool
3005             \int_gzero:N \c@tabularnote
3006             \@@_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).

```

3007     \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3008     {
3009         \tl_gput_right:Ne \g_@@_aux_tl
3010         {
3011             \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3012             { \int_use:N \g_@@_notes_caption_int }
3013         }
3014         \int_gzero:N \g_@@_notes_caption_int
3015     }
3016 }
3017 }

```

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

```

3018     \hbox
3019     {
3020         \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.

```

3021     \@@_create_extra_nodes:
3022     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3023 }

```

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 if compiles several times its tabular).

```

3024     \bool_lazy_any:nT
3025     {
3026         { ! \seq_if_empty_p:N \g_@@_notes_seq }
3027         { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3028         { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3029     }
3030     \@@_insert_tabularnotes:
3031     \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3032     \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3033     \end { minipage }
3034 }

```

```

3035 \cs_new_protected:Npn \@@_insert_caption:
3036 {
3037     \tl_if_empty:NF \l_@@_caption_tl
3038     {
3039         \cs_if_exist:NTF \c@caption
3040         { \@@_insert_caption_i: }
3041         { \@@_error:n { caption~outside~float } }
3042     }
3043 }

```

```

3044 \cs_new_protected:Npn \@@_insert_caption_i:
3045 {
3046     \group_begin:

```

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

```

3047     \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.

```

3048   \IfPackageLoadedT { floatrow }
3049   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3050   \tl_if_empty:NTF \l_@@_short_caption_tl
3051   { \caption }
3052   { \caption [ \l_@@_short_caption_tl ] }
3053   { \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.

```

3054   \bool_if:NF \g_@@_caption_finished_bool
3055   {
3056     \bool_gset_true:N \g_@@_caption_finished_bool
3057     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3058     \int_gzero:N \c@tabularnote
3059   }
3060   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3061   \group_end:
3062 }

3063 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3064 {
3065   \@@_error_or_warning:n { tabularnote-below-the-tabular }
3066   \@@_gredirect_none:n { tabularnote-below-the-tabular }
3067 }

3068 \cs_new_protected:Npn \@@_insert_tabularnotes:
3069 {
3070   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3071   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3072   \skip_vertical:N 0.65ex

```

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

```

3073   \group_begin:
3074   \l_@@_notes_code_before_tl
3075   \tl_if_empty:NF \g_@@_tabularnote_tl
3076   {
3077     \g_@@_tabularnote_tl \par
3078     \tl_gclear:N \g_@@_tabularnote_tl
3079   }

```

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.

```

3080   \int_compare:nNnT \c@tabularnote > \c_zero_int
3081   {
3082     \bool_if:NTF \l_@@_notes_para_bool
3083     {
3084       \begin { tabularnotes* }
3085         \seq_map_inline:Nn \g_@@_notes_seq
3086         { \@@_one_tabularnote:nm ##1 }
3087         \strut
3088         \end { tabularnotes* }

```

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

```

3089     \par
3090   }
3091   {
3092     \tabularnotes
3093     \seq_map_inline:Nn \g_@@_notes_seq

```

```

3094         { \@@_one_tabularnote:nn #1 }
3095         \strut
3096     \endtabularnotes
3097 }
3098 }
3099 \unskip
3100 \group_end:
3101 \bool_if:NT \l_@@_notes_bottomrule_bool
3102 {
3103     \IfPackageLoadedTF { booktabs }
3104     {

```

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

```

3105         \skip_vertical:N \aboverulesep

```

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

```

3106         { \CT@arc@ \hrule height \heavyrulewidth }
3107     }
3108     { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3109 }
3110 \l_@@_notes_code_after_tl
3111 \seq_gclear:N \g_@@_notes_seq
3112 \seq_gclear:N \g_@@_notes_in_caption_seq
3113 \int_gzero:N \c@tabularnote
3114 }

```

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

```

3115 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3116 {
3117     \tl_if_novalue:nTF { #1 }
3118     { \item }
3119     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3120 }

```

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`.

```

3121 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3122 {
3123     \pgfpicture
3124     \@@_qpoint:n { row - 1 }
3125     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3126     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3127     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3128     \endpgfpicture
3129     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3130     \int_if_zero:nT \l_@@_first_row_int
3131     {
3132         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3133         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3134     }
3135     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3136 }

```

Now, the general case.

```

3137 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3138 {

```

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

```

3139     \str_if_eq:eeT \l_@@_baseline_tl { t }
3140     { \cs_set_nopar:Npn \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`.

```

3141 \pgfpicture
3142 \@@_qpoint:n { row - 1 }
3143 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3144 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3145 {
3146   \int_set:Nn \l_tmpa_int
3147   {
3148     \str_range:Nnn
3149     \l_@@_baseline_tl
3150     6
3151     { \tl_count:o \l_@@_baseline_tl }
3152   }
3153   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3154 }
3155 {
3156   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3157   \bool_lazy_or:nnT
3158   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3159   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3160   {
3161     \@@_error:n { bad~value~for~baseline }
3162     \int_set:Nn \l_tmpa_int 1
3163   }
3164   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3165 }
3166 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3167 \endpgfpicture
3168 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3169 \int_if_zero:nT \l_@@_first_row_int
3170 {
3171   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3172   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3173 }
3174 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3175 }

```

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.

```

3176 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3177 {

```

We will compute the real width of both delimiters used.

```

3178   \dim_zero_new:N \l_@@_real_left_delim_dim
3179   \dim_zero_new:N \l_@@_real_right_delim_dim
3180   \hbox_set:Nn \l_tmpb_box
3181   {
3182     \c_math_toggle_token
3183     \left #1
3184     \vcenter
3185     {
3186       \vbox_to_ht:nn
3187       { \box_ht_plus_dp:N \l_tmpa_box }
3188       { }
3189     }
3190     \right .
3191     \c_math_toggle_token
3192   }
3193   \dim_set:Nn \l_@@_real_left_delim_dim
3194   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3195   \hbox_set:Nn \l_tmpb_box

```

```

3196     {
3197       \c_math_toggle_token
3198       \left .
3199       \vbox_to_ht:nn
3200         { \box_ht_plus_dp:N \l_tmpa_box }
3201         { }
3202       \right #2
3203       \c_math_toggle_token
3204     }
3205   \dim_set:Nn \l_@@_real_right_delim_dim
3206     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3207   \skip_horizontal:N \l_@@_left_delim_dim
3208   \skip_horizontal:N -\l_@@_real_left_delim_dim
3209   \@@_put_box_in_flow:
3210   \skip_horizontal:N \l_@@_right_delim_dim
3211   \skip_horizontal:N -\l_@@_real_right_delim_dim
3212 }

```

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).

```

3213 \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, an standard error will be raised by LaTeX for incorrect nested environments).

```

3214 {
3215   \peek_remove_spaces:n
3216   {
3217     \peek_meaning:NTF \end
3218     \@@_analyze_end:Nn
3219     {
3220       \@@_transform_preamble:

```

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

```

3221     \@@_array:o \g_@@_array_preamble_tl
3222   }
3223 }
3224 }
3225 {
3226   \@@_create_col_nodes:
3227   \endarray
3228 }

```

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

```

3229 \NewDocumentEnvironment { @@-light-syntax } { b }
3230 {

```

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`.

```

3231   \tl_if_empty:nT { #1 }
3232     { \@@_fatal:n { empty-environment } }
3233   \tl_if_in:nnT { #1 } { & }
3234     { \@@_fatal:n { ampersand-in-light-syntax } }
3235   \tl_if_in:nnT { #1 } { \ }
3236     { \@@_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`.

```
3237   \@@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```
3238   }
```

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.

```
3239   {
3240     \@@_create_col_nodes:
3241     \endarray
3242   }

3243   \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3244   {
3245     \tl_gput_right:Nn \g_nicematrix_code_after_tl { #2 }
```

The body of the array, which is stored in the argument `#1`, is now splitted into items (and *not* tokens).

```
3246     \seq_clear_new:N \l_@@_rows_seq
```

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

```
3247     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3248     \bool_if:NTF \l_@@_light_syntax_expanded_bool
3249       \seq_set_split:Nee
3250       \seq_set_split:Non
3251       \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

We delete the last row if it is empty.

```
3252     \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3253     \tl_if_empty:NF \l_tmpa_tl
3254     { \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.

```
3255     \int_compare:nNnT \l_@@_last_row_int = { -1 }
3256     { \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`.

```
3257     \tl_build_begin:N \l_@@_new_body_tl
3258     \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3259     \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3260     \@@_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).

```
3261     \seq_map_inline:Nn \l_@@_rows_seq
3262     {
3263       \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3264       \@@_line_with_light_syntax:n { ##1 }
3265     }
3266     \tl_build_end:N \l_@@_new_body_tl

3267     \int_compare:nNnT \l_@@_last_col_int = { -1 }
3268     {
3269       \int_set:Nn \l_@@_last_col_int
3270       { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3271     }
```

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.

```
3272 \@@_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`).

```
3273 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3274 }
3275 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }
3276 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3277 {
3278   \seq_clear_new:N \l_@@_cells_seq
3279   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3280   \int_set:Nn \l_@@_nb_cols_int
3281   {
3282     \int_max:nn
3283     \l_@@_nb_cols_int
3284     { \seq_count:N \l_@@_cells_seq }
3285   }
3286   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3287   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3288   \seq_map_inline:Nn \l_@@_cells_seq
3289   { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3290 }
```

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`.

```
3291 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3292 {
3293   \str_if_eq:eeT \g_@@_name_env_str { #2 }
3294   { \@@_fatal:n { empty-environment } }
```

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

```
3295   \end { #2 }
3296 }
```

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`).

```
3297 \cs_new:Npn \@@_create_col_nodes:
3298 {
3299   \crrc
3300   \int_if_zero:nT \l_@@_first_col_int
3301   {
3302     \omit
3303     \hbox_overlap_left:n
3304     {
3305       \bool_if:NT \l_@@_code_before_bool
3306       { \pgfsys@markposition { \@@_env: - col - 0 } }
3307       \pgfpicture
3308       \pgfrememberpicturepositiononpagetrue
3309       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3310       \str_if_empty:NF \l_@@_name_str
3311       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3312       \endpgfpicture
3313       \skip_horizontal:N 2\col@sep
3314       \skip_horizontal:N \g_@@_width_first_col_dim
3315     }
3316 }
```

```

3316     &
3317   }
3318   \omit

```

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

```

3319   \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`).

```

3320   \int_if_zero:nTF \l_@@_first_col_int
3321   {
3322     \bool_if:NT \l_@@_code_before_bool
3323     {
3324       \hbox
3325       {
3326         \skip_horizontal:N -0.5\arrayrulewidth
3327         \pgfsys@markposition { \@@_env: - col - 1 }
3328         \skip_horizontal:N 0.5\arrayrulewidth
3329       }
3330     }
3331     \pgfpicture
3332     \pgfrememberpicturepositiononpagetrue
3333     \pgfcoordinate { \@@_env: - col - 1 }
3334     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3335     \str_if_empty:NF \l_@@_name_str
3336     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3337     \endpgfpicture
3338   }
3339   {
3340     \bool_if:NT \l_@@_code_before_bool
3341     {
3342       \hbox
3343       {
3344         \skip_horizontal:N 0.5\arrayrulewidth
3345         \pgfsys@markposition { \@@_env: - col - 1 }
3346         \skip_horizontal:N -0.5\arrayrulewidth
3347       }
3348     }
3349     \pgfpicture
3350     \pgfrememberpicturepositiononpagetrue
3351     \pgfcoordinate { \@@_env: - col - 1 }
3352     { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3353     \str_if_empty:NF \l_@@_name_str
3354     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3355     \endpgfpicture
3356   }

```

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.

```

3357   \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3358   \bool_if:NF \l_@@_auto_columns_width_bool
3359   { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3360   {
3361     \bool_lazy_and:nnTF
3362     \l_@@_auto_columns_width_bool
3363     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3364     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3365     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3366     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3367   }

```

```

3368 \skip_horizontal:N \g_tmpa_skip
3369 \hbox
3370 {
3371   \bool_if:NT \l_@@_code_before_bool
3372   {
3373     \hbox
3374     {
3375       \skip_horizontal:N -0.5\arrayrulewidth
3376       \pgfsys@markposition { \@@_env: - col - 2 }
3377       \skip_horizontal:N 0.5\arrayrulewidth
3378     }
3379   }
3380   \pgfpicture
3381   \pgfrememberpicturepositiononpagetrue
3382   \pgfcoordinate { \@@_env: - col - 2 }
3383   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3384   \str_if_empty:NF \l_@@_name_str
3385   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3386   \endpgfpicture
3387 }

```

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.

```

3388 \int_gset_eq:NN \g_tmpa_int \c_one_int
3389 \bool_if:NTF \g_@@_last_col_found_bool
3390 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3391 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3392 {
3393   &
3394   \omit
3395   \int_gincr:N \g_tmpa_int

```

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

```

3396 \skip_horizontal:N \g_tmpa_skip
3397 \bool_if:NT \l_@@_code_before_bool
3398 {
3399   \hbox
3400   {
3401     \skip_horizontal:N -0.5\arrayrulewidth
3402     \pgfsys@markposition
3403     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3404     \skip_horizontal:N 0.5\arrayrulewidth
3405   }
3406 }

```

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

```

3407 \pgfpicture
3408 \pgfrememberpicturepositiononpagetrue
3409 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3410 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3411 \str_if_empty:NF \l_@@_name_str
3412 {
3413   \pgfnodealias
3414   { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3415   { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3416 }
3417 \endpgfpicture
3418 }

3419 &
3420 \omit

```

The two following lines have been added on 2021-12-15 to solve a bug mentioned by Joao Luis Soares by mail.

```

3421 \int_if_zero:nT \g_@@_col_total_int
3422   { \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill } }
3423 \skip_horizontal:N \g_tmpa_skip
3424 \int_gincr:N \g_tmpa_int
3425 \bool_lazy_any:nF
3426   {
3427     \g_@@_delims_bool
3428     \l_@@_tabular_bool
3429     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3430     \l_@@_exterior_arraycolsep_bool
3431     \l_@@_bar_at_end_of_pream_bool
3432   }
3433   { \skip_horizontal:N -\col@sep }
3434 \bool_if:NT \l_@@_code_before_bool
3435   {
3436     \hbox
3437     {
3438       \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.

```

3439     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3440     { \skip_horizontal:N -\arraycolsep }
3441     \pgfsys@markposition
3442     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3443     \skip_horizontal:N 0.5\arrayrulewidth
3444     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3445     { \skip_horizontal:N \arraycolsep }
3446   }
3447 }
3448 \pgfpicture
3449 \pgfrememberpicturepositiononpagetrue
3450 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3451 {
3452   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3453   {
3454     \pgfpoint
3455     { - 0.5 \arrayrulewidth - \arraycolsep }
3456     \c_zero_dim
3457   }
3458   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3459 }
3460 \str_if_empty:NF \l_@@_name_str
3461 {
3462   \pgfnodealias
3463   { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3464   { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3465 }
3466 \endpgfpicture

3467 \bool_if:NT \g_@@_last_col_found_bool
3468 {
3469   \hbox_overlap_right:n
3470   {
3471     \skip_horizontal:N \g_@@_width_last_col_dim
3472     \skip_horizontal:N \col@sep
3473     \bool_if:NT \l_@@_code_before_bool
3474     {
3475       \pgfsys@markposition
3476       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3477     }
3478   }

```

```

3479     \pgfrememberpicturepositiononpagetrue
3480     \pgfcoordinate
3481     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3482     \pgfpointorigin
3483     \str_if_empty:NF \l_@@_name_str
3484     {
3485         \pgfnodealias
3486         {
3487             \l_@@_name_str - col
3488             - \int_eval:n { \g_@@_col_total_int + 1 }
3489         }
3490         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3491     }
3492     \endpgfpicture
3493 }
3494 }
3495 % \cr
3496 }

```

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

```

3497 \tl_const:Nn \c_@@_preamble_first_col_tl
3498 {
3499     >
3500     {

```

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

```

3501     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3502     \bool_gset_true:N \g_@@_after_col_zero_bool
3503     \@@_begin_of_row:

```

The contents of the cell is constructed in the box `\l_@@_cell_box` because we have to compute some dimensions of this box.

```

3504     \hbox_set:Nw \l_@@_cell_box
3505     \@@_math_toggle:
3506     \@@_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”.

```

3507     \int_compare:nNnT \c@iRow > \c_zero_int
3508     {
3509         \bool_lazy_or:nnT
3510         { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3511         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3512         {
3513             \l_@@_code_for_first_col_tl
3514             \xglobal \colorlet { nicematrix-first-col } { . }
3515         }
3516     }
3517 }

```

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`.

```

3518     l
3519     <
3520     {
3521         \@@_math_toggle:
3522         \hbox_set_end:
3523         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3524         \@@_adjust_size_box:
3525         \@@_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.

```

3526 \dim_gset:Nn \g_@@_width_first_col_dim
3527 { \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.

```

3528 \hbox_overlap_left:n
3529 {
3530   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3531   \@@_node_for_cell:
3532   { \box_use_drop:N \l_@@_cell_box }
3533   \skip_horizontal:N \l_@@_left_delim_dim
3534   \skip_horizontal:N \l_@@_left_margin_dim
3535   \skip_horizontal:N \l_@@_extra_left_margin_dim
3536 }
3537 \bool_gset_false:N \g_@@_empty_cell_bool
3538 \skip_horizontal:N -2\col@sep
3539 }
3540 }

```

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

```

3541 \tl_const:Nn \c_@@_preamble_last_col_tl
3542 {
3543   >
3544   {
3545     \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).

```

3546 \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.

```

3547 \bool_gset_true:N \g_@@_last_col_found_bool
3548 \int_gincr:N \c@jCol
3549 \int_gset_eq:NN \g_@@_col_total_int \c@jCol

```

The contents of the cell is constructed in the box `\l_tmpa_box` because we have to compute some dimensions of this box.

```

3550 \hbox_set:Nw \l_@@_cell_box
3551 \@@_math_toggle:
3552 \@@_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”.

```

3553 \int_compare:nNnT \c@iRow > \c_zero_int
3554 {
3555   \bool_lazy_or:nnT
3556   { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3557   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3558   {
3559     \l_@@_code_for_last_col_tl
3560     \xglobal \colorlet { nicematrix-last-col } { . }
3561   }
3562 }
3563 }
3564 1
3565 <
3566 {
3567   \@@_math_toggle:
3568   \hbox_set_end:
3569   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3570   \@@_adjust_size_box:
3571   \@@_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.

```

3572 \dim_gset:Nn \g_@@_width_last_col_dim
3573 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3574 \skip_horizontal:N -2\col@sep

```

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

```

3575 \hbox_overlap_right:n
3576 {
3577   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3578   {
3579     \skip_horizontal:N \l_@@_right_delim_dim
3580     \skip_horizontal:N \l_@@_right_margin_dim
3581     \skip_horizontal:N \l_@@_extra_right_margin_dim
3582     \@@_node_for_cell:
3583   }
3584 }
3585 \bool_gset_false:N \g_@@_empty_cell_bool
3586 }
3587 }

```

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

```

3588 \NewDocumentEnvironment { NiceArray } { }
3589 {
3590   \bool_gset_false:N \g_@@_delims_bool
3591   \str_if_empty:NT \g_@@_name_env_str
3592   { \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).

```

3593   \NiceArrayWithDelims . .
3594 }
3595 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

3596 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3597 {
3598   \NewDocumentEnvironment { #1 NiceArray } { }
3599   {
3600     \bool_gset_true:N \g_@@_delims_bool
3601     \str_if_empty:NT \g_@@_name_env_str
3602     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3603     \@@_test_if_math_mode:
3604     \NiceArrayWithDelims #2 #3
3605   }
3606   { \endNiceArrayWithDelims }
3607 }
3608 \@@_def_env:nnn p ( )
3609 \@@_def_env:nnn b [ ]
3610 \@@_def_env:nnn B \{ \}
3611 \@@_def_env:nnn v | |
3612 \@@_def_env:nnn V \| \|

```

## 13 The environment `{NiceMatrix}` and its variants

```

3613 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3614 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3615 {
3616   \bool_set_false:N \l_@@_preamble_bool
3617   \tl_clear:N \l_tmpa_tl
3618   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3619     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3620   \tl_put_right:Nn \l_tmpa_tl
3621     {
3622       *
3623       {
3624         \int_case:nnF \l_@@_last_col_int
3625           {
3626             { -2 } { \c@MaxMatrixCols }
3627             { -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).

```

3628       }
3629       { \int_eval:n { \l_@@_last_col_int - 1 } }
3630     }
3631     { #2 }
3632   }
3633   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3634   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3635 }
3636 \clist_map_inline:nn { p , b , B , v , V }
3637 {
3638   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
3639   {
3640     \bool_gset_true:N \g_@@_delims_bool
3641     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3642     \int_if_zero:nT \l_@@_last_col_int
3643     {
3644       \bool_set_true:N \l_@@_last_col_without_value_bool
3645       \int_set:Nn \l_@@_last_col_int { -1 }
3646     }
3647     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3648     \@@_begin_of_NiceMatrix:no { #1 } \l_@@_columns_type_tl
3649   }
3650   { \use:c { end #1 NiceArray } }
3651 }

```

We define also an environment `{NiceMatrix}`

```

3652 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3653 {
3654   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3655   \int_if_zero:nT \l_@@_last_col_int
3656   {
3657     \bool_set_true:N \l_@@_last_col_without_value_bool
3658     \int_set:Nn \l_@@_last_col_int { -1 }
3659   }
3660   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3661   \bool_lazy_or:nnT
3662     { \clist_if_empty_p:N \l_@@_vlines_clist }
3663     { \l_@@_except_borders_bool }
3664     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3665   \@@_begin_of_NiceMatrix:no { } \l_@@_columns_type_tl
3666 }
3667 { \endNiceArray }

```

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

```
3668 \cs_new_protected:Npn \@@_NotEmpty:
3669 { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

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

```
3670 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3671 {
```

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`.

```
3672   \dim_compare:nNt \l_@@_width_dim = \c_zero_dim
3673     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3674   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3675   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3676   \tl_if_empty:NF \l_@@_short_caption_tl
3677     {
3678     \tl_if_empty:NT \l_@@_caption_tl
3679       {
3680         \@@_error_or_warning:n { short-caption-without-caption }
3681         \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3682       }
3683     }
3684   \tl_if_empty:NF \l_@@_label_tl
3685     {
3686     \tl_if_empty:NT \l_@@_caption_tl
3687       { \@@_error_or_warning:n { label-without-caption } }
3688     }
3689   \NewDocumentEnvironment { TabularNote } { b }
3690     {
3691     \bool_if:NTF \l_@@_in_code_after_bool
3692       { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3693       {
3694         \tl_if_empty:NF \g_@@_tabularnote_tl
3695           { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3696         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3697       }
3698     }
3699     { }
3700   \@@_settings_for_tabular:
3701   \NiceArray { #2 }
3702   }
3703   {
3704   \endNiceArray
3705   \bool_if:NT \c_@@_testphase_table_bool
3706     { \UseTaggingSocket { tbl / hmode / end } }
3707   }
3708 \cs_new_protected:Npn \@@_settings_for_tabular:
3709 {
3710   \bool_set_true:N \l_@@_tabular_bool
3711   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3712   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3713   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3714 }

3715 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }
3716 {
3717   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3718   \dim_zero_new:N \l_@@_width_dim
3719   \dim_set:Nn \l_@@_width_dim { #1 }
3720   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3721   \@@_settings_for_tabular:
```

```

3722 \NiceArray { #3 }
3723 }
3724 {
3725 \endNiceArray
3726 \int_if_zero:nT \g_@@_total_X_weight_int
3727 { \@@_error:n { NiceTabularX~without~X } }
3728 }

3729 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3730 {
3731 \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3732 \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3733 \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3734 \@@_settings_for_tabular:
3735 \NiceArray { #3 }
3736 }
3737 { \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`).

```

3738 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3739 {
3740 \bool_lazy_all:nT
3741 {
3742 { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3743 \l_@@_hvlines_bool
3744 { ! \g_@@_delims_bool }
3745 { ! \l_@@_except_borders_bool }
3746 }
3747 {
3748 \bool_set_true:N \l_@@_except_borders_bool
3749 \clist_if_empty:NF \l_@@_corners_clist
3750 { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3751 \tl_gput_right:Nn \g_@@_pre_code_after_tl
3752 {
3753 \@@_stroke_block:nnn
3754 {
3755 rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3756 draw = \l_@@_rules_color_tl
3757 }
3758 { 1-1 }
3759 { \int_use:N \c@iRow - \int_use:N \c@jCol }
3760 }
3761 }
3762 }

3763 \cs_new_protected:Npn \@@_after_array:
3764 {

```

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`.

```

3765 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3766 \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.

```
3767 \bool_if:NT \g_@@_last_col_found_bool
3768 { \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`.

```
3769 \bool_if:NT \l_@@_last_col_without_value_bool
3770 { \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.

```
3771 \bool_if:NT \l_@@_last_row_without_value_bool
3772 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }
```

```
3773 \tl_gput_right:Ne \g_@@_aux_tl
3774 {
3775   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3776   {
3777     \int_use:N \l_@@_first_row_int ,
3778     \int_use:N \c@iRow ,
3779     \int_use:N \g_@@_row_total_int ,
3780     \int_use:N \l_@@_first_col_int ,
3781     \int_use:N \c@jCol ,
3782     \int_use:N \g_@@_col_total_int
3783   }
3784 }
```

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`).

```
3785 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3786 {
3787   \tl_gput_right:Ne \g_@@_aux_tl
3788   {
3789     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3790     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3791   }
3792 }
3793 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3794 {
3795   \tl_gput_right:Ne \g_@@_aux_tl
3796   {
3797     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3798     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3799     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3800     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3801   }
3802 }
```

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

```
3803 \@@_create_diag_nodes:
```

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

```
3804 \pgfpicture
3805 \int_step_inline:nn \c@iRow
3806 {
3807   \pgfnodealias
3808   { \@@_env: - ##1 - last }
3809   { \@@_env: - ##1 - \int_use:N \c@jCol }
```

```

3810     }
3811     \int_step_inline:nn \c@jCol
3812     {
3813         \pgfnodealias
3814         { \@@_env: - last - ##1 }
3815         { \@@_env: - \int_use:N \c@iRow - ##1 }
3816     }
3817     \str_if_empty:NF \l_@@_name_str
3818     {
3819         \int_step_inline:nn \c@iRow
3820         {
3821             \pgfnodealias
3822             { \l_@@_name_str - ##1 - last }
3823             { \@@_env: - ##1 - \int_use:N \c@jCol }
3824         }
3825         \int_step_inline:nn \c@jCol
3826         {
3827             \pgfnodealias
3828             { \l_@@_name_str - last - ##1 }
3829             { \@@_env: - \int_use:N \c@iRow - ##1 }
3830         }
3831     }
3832     \endpgfpicture

```

By default, the diagonal lines will be parallelized<sup>11</sup>. 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.

```

3833     \bool_if:NT \l_@@_parallelize_diags_bool
3834     {
3835         \int_gzero_new:N \g_@@_ddots_int
3836         \int_gzero_new:N \g_@@_iddots_int

```

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the  $\Delta_x$  and  $\Delta_y$  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  $\Delta_x$  and  $\Delta_y$  of the first `\Iddots` diagonal.

```

3837         \dim_gzero_new:N \g_@@_delta_x_one_dim
3838         \dim_gzero_new:N \g_@@_delta_y_one_dim
3839         \dim_gzero_new:N \g_@@_delta_x_two_dim
3840         \dim_gzero_new:N \g_@@_delta_y_two_dim
3841     }
3842     \int_zero_new:N \l_@@_initial_i_int
3843     \int_zero_new:N \l_@@_initial_j_int
3844     \int_zero_new:N \l_@@_final_i_int
3845     \int_zero_new:N \l_@@_final_j_int
3846     \bool_set_false:N \l_@@_initial_open_bool
3847     \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.

```

3848     \bool_if:NT \l_@@_small_bool
3849     {
3850         \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3851         \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.

```

3852         \dim_set:Nn \l_@@_xdots_shorten_start_dim
3853         { 0.6 \l_@@_xdots_shorten_start_dim }
3854         \dim_set:Nn \l_@@_xdots_shorten_end_dim

```

<sup>11</sup>It's possible to use the option `parallelize-diags` to disable this parallelization.

```

3855     { 0.6 \l_@@_xdots_shorten_end_dim }
3856   }

```

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

```

3857   \@@_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.

```

3858   \clist_if_empty:NF \l_@@_corners_clist \@@_compute_corners:

```

The sequence `\g_@@_pos_of_blocks_seq` must be “adjusted” (for the case where the user have written something like `\Block{1-*}`).

```

3859   \@@_adjust_pos_of_blocks_seq:
3860   \@@_deal_with_rounded_corners:
3861   \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3862   \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3863   \IfPackageLoadedT { tikz }
3864   {
3865     \tikzset
3866     {
3867       every-picture / .style =
3868       {
3869         overlay ,
3870         remember~picture ,
3871         name~prefix = \@@_env: -
3872       }
3873     }
3874   }
3875   \bool_if:NT \c_@@_tagging_array_bool
3876   { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3877   \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3878   \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3879   \cs_set_eq:NN \OverBrace \@@_OverBrace
3880   \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3881   \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3882   \cs_set_eq:NN \line \@@_line
3883   \g_@@_pre_code_after_tl
3884   \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 `\Code-after` to be *no-op* now.

```

3885   \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).

```

3886   \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`).

```

3887   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3888   { \@@_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:`.

```

3889   \bool_set_true:N \l_@@_in_code_after_bool
3890   \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3891   \scan_stop:
3892   \tl_gclear:N \g_nicematrix_code_after_tl
3893   \group_end:

```

`\g_@@_pre_code_before_tl` is for instructions in the cells of the array such as `\rowcolor` and `\cellcolor` (when the key `color-inside` is in force). These instructions will be written on the aux file to be added to the code-before in the next run.

```

3894   \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3895   \tl_if_empty:NF \g_@@_pre_code_before_tl
3896   {
3897     \tl_gput_right:Ne \g_@@_aux_tl
3898     {
3899       \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3900       { \exp_not:o \g_@@_pre_code_before_tl }
3901     }
3902     \tl_gclear:N \g_@@_pre_code_before_tl
3903   }
3904   \tl_if_empty:NF \g_nicematrix_code_before_tl
3905   {
3906     \tl_gput_right:Ne \g_@@_aux_tl
3907     {
3908       \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3909       { \exp_not:o \g_nicematrix_code_before_tl }
3910     }
3911     \tl_gclear:N \g_nicematrix_code_before_tl
3912   }

3913   \str_gclear:N \g_@@_name_env_str
3914   \@@_restore_iRow_jCol:

```

The command `\CT@arc@` contains the instruction of color for the rules of the array<sup>12</sup>. 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`.

```

3915   \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3916   }

```

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`.

```

3917 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3918 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

```

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).

```

3919 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3920 {

```

---

<sup>12</sup>e.g. `\color[rgb]{0.5,0.5,0}`

```

3921 \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3922 { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3923 }

```

The following command must *not* be protected.

```

3924 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3925 {
3926   { #1 }
3927   { #2 }
3928   {
3929     \int_compare:nNnTF { #3 } > { 99 }
3930     { \int_use:N \c@iRow }
3931     { #3 }
3932   }
3933   {
3934     \int_compare:nNnTF { #4 } > { 99 }
3935     { \int_use:N \c@jCol }
3936     { #4 }
3937   }
3938   { #5 }
3939 }

```

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).

```

3940 \hook_gput_code:nnn { begindocument } { . }
3941 {
3942   \cs_new_protected:Npe \@@_draw_dotted_lines:
3943   {
3944     \c_@@_pgfortikzpicture_tl
3945     \@@_draw_dotted_lines_i:
3946     \c_@@_endpgfortikzpicture_tl
3947   }
3948 }

```

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

```

3949 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3950 {
3951   \pgfrememberpicturepositiononpagetrue
3952   \pgf@relevantforpicturesizefalse
3953   \g_@@_HVdotsfor_lines_tl
3954   \g_@@_Vdots_lines_tl
3955   \g_@@_Ddots_lines_tl
3956   \g_@@_Iddots_lines_tl
3957   \g_@@_Cdots_lines_tl
3958   \g_@@_Ldots_lines_tl
3959 }

```

```

3960 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3961 {
3962   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3963   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3964 }

```

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

```

3965 \pgfdeclareshape { @@_diag_node }
3966 {
3967   \savedanchor { \five }
3968   {
3969     \dim_gset_eq:NN \pgf@x \l_tmpa_dim

```

```

3970     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3971   }
3972   \anchor { 5 } { \five }
3973   \anchor { center } { \pgfpointorigin }
3974   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
3975   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
3976   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
3977   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
3978   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
3979   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
3980   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
3981   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
3982   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
3983   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
3984 }

```

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).

```

3985 \cs_new_protected:Npn \@@_create_diag_nodes:
3986 {
3987   \pgfpicture
3988   \pgfrememberpicturepositiononpagetrue
3989   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3990   {
3991     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3992     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3993     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3994     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3995     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3996     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3997     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3998     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3999     \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.

```

4000     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4001     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4002     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4003     \str_if_empty:NF \l_@@_name_str
4004     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4005   }

```

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

```

4006     \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4007     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4008     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4009     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4010     \pgfcoordinate
4011     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4012     \pgfnodealias
4013     { \@@_env: - last }
4014     { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4015     \str_if_empty:NF \l_@@_name_str
4016     {
4017       \pgfnodealias
4018       { \l_@@_name_str - \int_use:N \l_tmpa_int }
4019       { \@@_env: - \int_use:N \l_tmpa_int }
4020       \pgfnodealias
4021       { \l_@@_name_str - last }
4022       { \@@_env: - last }
4023     }

```

```
4024 \endpgfpicture
4025 }
```

## 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.

$$\left( \begin{array}{ccc} a + b + c & a + b & a \\ a \dots\dots\dots & & \\ a & a + b & a + b + c \end{array} \right)$$

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.

```
4026 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4027 {
```

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

```
4028 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4029 \int_set:Nn \l_@@_initial_i_int { #1 }
4030 \int_set:Nn \l_@@_initial_j_int { #2 }
4031 \int_set:Nn \l_@@_final_i_int { #1 }
4032 \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.

```
4033 \bool_set_false:N \l_@@_stop_loop_bool
4034 \bool_do_until:Nn \l_@@_stop_loop_bool
4035 {
4036 \int_add:Nn \l_@@_final_i_int { #3 }
4037 \int_add:Nn \l_@@_final_j_int { #4 }
4038 \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.

```

4039     \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4040     \if_int_compare:w #3 = \c_one_int
4041     \bool_set_true:N \l_@@_final_open_bool
4042     \else:
4043     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4044     \bool_set_true:N \l_@@_final_open_bool
4045     \fi:
4046     \fi:
4047     \else:
4048     \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4049     \if_int_compare:w #4 = -1
4050     \bool_set_true:N \l_@@_final_open_bool
4051     \fi:
4052     \else:
4053     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4054     \if_int_compare:w #4 = \c_one_int
4055     \bool_set_true:N \l_@@_final_open_bool
4056     \fi:
4057     \fi:
4058     \fi:
4059     \fi:
4060     \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.

```

4061     {

```

We do a step backwards.

```

4062         \int_sub:Nn \l_@@_final_i_int { #3 }
4063         \int_sub:Nn \l_@@_final_j_int { #4 }
4064         \bool_set_true:N \l_@@_stop_loop_bool
4065     }

```

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`.

```

4066     {
4067     \cs_if_exist:cTF
4068     {
4069     @@ _ dotted _
4070     \int_use:N \l_@@_final_i_int -
4071     \int_use:N \l_@@_final_j_int
4072     }
4073     {
4074     \int_sub:Nn \l_@@_final_i_int { #3 }
4075     \int_sub:Nn \l_@@_final_j_int { #4 }
4076     \bool_set_true:N \l_@@_final_open_bool
4077     \bool_set_true:N \l_@@_stop_loop_bool
4078     }
4079     {
4080     \cs_if_exist:cTF
4081     {
4082     pgf @ sh @ ns @ \@@_env:
4083     - \int_use:N \l_@@_final_i_int
4084     - \int_use:N \l_@@_final_j_int
4085     }
4086     { \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.

```

4087     {

```

```

4088         \cs_set_nopar:cpn
4089         {
4090             @@ _ dotted _
4091             \int_use:N \l_@@_final_i_int -
4092             \int_use:N \l_@@_final_j_int
4093         }
4094         { }
4095     }
4096 }
4097 }
4098 }

```

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

```

4099     \bool_set_false:N \l_@@_stop_loop_bool

```

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

```

4100     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4101     \bool_do_until:Nn \l_@@_stop_loop_bool
4102     {
4103         \int_sub:Nn \l_@@_initial_i_int { #3 }
4104         \int_sub:Nn \l_@@_initial_j_int { #4 }
4105         \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.

```

4106         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4107         \if_int_compare:w #3 = \c_one_int
4108             \bool_set_true:N \l_@@_initial_open_bool
4109         \else:

```

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

```

4110             \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4111                 \bool_set_true:N \l_@@_initial_open_bool
4112             \fi:
4113         \fi:
4114     \else:
4115         \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4116             \if_int_compare:w #4 = \c_one_int
4117                 \bool_set_true:N \l_@@_initial_open_bool
4118             \fi:
4119         \else:
4120             \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4121                 \if_int_compare:w #4 = -1
4122                     \bool_set_true:N \l_@@_initial_open_bool
4123                 \fi:
4124             \fi:
4125         \fi:
4126     \fi:
4127     \bool_if:NTF \l_@@_initial_open_bool
4128     {
4129         \int_add:Nn \l_@@_initial_i_int { #3 }
4130         \int_add:Nn \l_@@_initial_j_int { #4 }
4131         \bool_set_true:N \l_@@_stop_loop_bool
4132     }
4133     {
4134         \cs_if_exist:cTF
4135         {
4136             @@ _ dotted _
4137             \int_use:N \l_@@_initial_i_int -
4138             \int_use:N \l_@@_initial_j_int
4139         }

```

```

4140     {
4141         \int_add:Nn \l_@@_initial_i_int { #3 }
4142         \int_add:Nn \l_@@_initial_j_int { #4 }
4143         \bool_set_true:N \l_@@_initial_open_bool
4144         \bool_set_true:N \l_@@_stop_loop_bool
4145     }
4146     {
4147         \cs_if_exist:cTF
4148         {
4149             pgf @ sh @ ns @ \@@_env:
4150             - \int_use:N \l_@@_initial_i_int
4151             - \int_use:N \l_@@_initial_j_int
4152         }
4153         { \bool_set_true:N \l_@@_stop_loop_bool }
4154         {
4155             \cs_set_nopar:cpn
4156             {
4157                 @@ _ dotted _
4158                 \int_use:N \l_@@_initial_i_int -
4159                 \int_use:N \l_@@_initial_j_int
4160             }
4161             { }
4162         }
4163     }
4164 }
4165 }

```

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.

```

4166     \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4167     {
4168         { \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`.

```

4169         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4170         { \int_use:N \l_@@_final_i_int }
4171         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4172         { } % for the name of the block
4173     }
4174 }

```

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.

```

4175     \cs_new_protected:Npn \@@_open_shorten:
4176     {
4177         \bool_if:NT \l_@@_initial_open_bool
4178         { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4179         \bool_if:NT \l_@@_final_open_bool
4180         { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4181     }

```

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).

```

4182     \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4183     {
4184         \int_set_eq:NN \l_@@_row_min_int \c_one_int

```

```

4185 \int_set_eq:NN \l_@@_col_min_int \c_one_int
4186 \int_set_eq:NN \l_@@_row_max_int \c@iRow
4187 \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`.

```

4188 \seq_if_empty:NF \g_@@_submatrix_seq
4189 {
4190   \seq_map_inline:Nn \g_@@_submatrix_seq
4191     { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
4192 }
4193 }

```

`#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:nnnnn #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.

```

4194 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnn #1 #2 #3 #4 #5 #6
4195 {
4196   \if_int_compare:w #3 > #1
4197   \else:
4198     \if_int_compare:w #1 > #5
4199     \else:
4200       \if_int_compare:w #4 > #2
4201       \else:
4202         \if_int_compare:w #2 > #6
4203         \else:
4204           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4205           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4206           \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4207           \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4208         \fi:
4209       \fi:
4210     \fi:
4211   \fi:
4212 }

4213 \cs_new_protected:Npn \@@_set_initial_coords:
4214 {
4215   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4216   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4217 }
4218 \cs_new_protected:Npn \@@_set_final_coords:
4219 {

```



```

4220 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4221 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4222 }
4223 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4224 {
4225 \pgfpointanchor
4226 {
4227 \@@_env:
4228 - \int_use:N \l_@@_initial_i_int
4229 - \int_use:N \l_@@_initial_j_int
4230 }
4231 { #1 }
4232 \@@_set_initial_coords:
4233 }
4234 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4235 {
4236 \pgfpointanchor
4237 {
4238 \@@_env:
4239 - \int_use:N \l_@@_final_i_int
4240 - \int_use:N \l_@@_final_j_int
4241 }
4242 { #1 }
4243 \@@_set_final_coords:
4244 }
4245 \cs_new_protected:Npn \@@_open_x_initial_dim:
4246 {
4247 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4248 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4249 {
4250 \cs_if_exist:cT
4251 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4252 {
4253 \pgfpointanchor
4254 { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4255 { west }
4256 \dim_set:Nn \l_@@_x_initial_dim
4257 { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4258 }
4259 }

```

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

```

4260 \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4261 {
4262 \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4263 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4264 \dim_add:Nn \l_@@_x_initial_dim \col@sep
4265 }
4266 }
4267 \cs_new_protected:Npn \@@_open_x_final_dim:
4268 {
4269 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4270 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4271 {
4272 \cs_if_exist:cT
4273 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4274 {
4275 \pgfpointanchor
4276 { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4277 { east }
4278 \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4279 { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4280 }

```

```
4281     }
```

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

```
4282     \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4283     {
4284         \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4285         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4286         \dim_sub:Nn \l_@@_x_final_dim \col@sep
4287     }
4288 }
```

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.

```
4289 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4290 {
4291     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4292     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4293     {
4294         \@@_find_extremities_of_line:nmmm { #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.

```
4295         \group_begin:
4296         \@@_open_shorten:
4297         \int_if_zero:nTF { #1 }
4298         { \color { nicematrix-first-row } }
4299         {
```

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.

```
4300             \int_compare:nNnT { #1 } = \l_@@_last_row_int
4301             { \color { nicematrix-last-row } }
4302         }
4303         \keys_set:nn { nicematrix / xdots } { #3 }
4304         \@@_color:o \l_@@_xdots_color_tl
4305         \@@_actually_draw_Ldots:
4306     \group_end:
4307 }
4308 }
```

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`.

```
4309 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4310 {
4311     \bool_if:NTF \l_@@_initial_open_bool
4312     {
4313         \@@_open_x_initial_dim:
4314         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4315         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4316     }
4317     { \@@_set_initial_coords_from_anchor:n { base-east } }
```

```

4318 \bool_if:NTF \l_@@_final_open_bool
4319 {
4320   \@@_open_x_final_dim:
4321   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4322   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4323 }
4324 { \@@_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.

```

4325 \bool_lazy_all:nTF
4326 {
4327   \l_@@_initial_open_bool
4328   \l_@@_final_open_bool
4329   { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4330 }
4331 {
4332   \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4333   \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4334 }

```

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 (?).

```

4335 {
4336   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4337   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4338 }
4339 \@@_draw_line:
4340 }

```

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.

```

4341 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4342 {
4343   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4344   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4345   {
4346     \@@_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.

```

4347   \group_begin:
4348   \@@_open_shorten:
4349   \int_if_zero:nTF { #1 }
4350   { \color { nicematrix-first-row } }
4351   {

```

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.

```

4352     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4353     { \color { nicematrix-last-row } }
4354   }
4355   \keys_set:nn { nicematrix / xdots } { #3 }
4356   \@@_color:o \l_@@_xdots_color_tl
4357   \@@_actually_draw_Cdots:
4358   \group_end:
4359 }
4360 }

```

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.

```

4361 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4362 {
4363   \bool_if:NTF \l_@@_initial_open_bool
4364     { \@@_open_x_initial_dim: }
4365     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4366   \bool_if:NTF \l_@@_final_open_bool
4367     { \@@_open_x_final_dim: }
4368     { \@@_set_final_coords_from_anchor:n { mid-west } }
4369   \bool_lazy_and:mnTF
4370     \l_@@_initial_open_bool
4371     \l_@@_final_open_bool
4372   {
4373     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4374     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4375     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4376     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4377     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4378   }
4379   {
4380     \bool_if:NT \l_@@_initial_open_bool
4381       { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4382     \bool_if:NT \l_@@_final_open_bool
4383       { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4384   }
4385   \@@_draw_line:
4386 }

4387 \cs_new_protected:Npn \@@_open_y_initial_dim:
4388 {
4389   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4390   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4391     {
4392       \cs_if_exist:cT
4393         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4394         {
4395           \pgfpointanchor
4396             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4397             { north }
4398           \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4399             { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4400         }
4401     }
4402   \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4403     {
4404       \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4405       \dim_set:Nn \l_@@_y_initial_dim
4406         {
4407           \fp_to_dim:n
4408             {
4409               \pgf@y
4410               + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4411             }
4412         }
4413     }
4414 }

```

```

4415 \cs_new_protected:Npn \l_@@_open_y_final_dim:
4416 {
4417   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4418   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4419   {
4420     \cs_if_exist:cT
4421     { \pgf @ sh @ ns @ \l_@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4422     {
4423       \pgfpointanchor
4424       { \l_@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4425       { south }
4426       \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4427       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4428     }
4429   }
4430   \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4431   {
4432     \l_@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4433     \dim_set:Nn \l_@@_y_final_dim
4434     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4435   }
4436 }

```

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.

```

4437 \cs_new_protected:Npn \l_@@_draw_Vdots:nnn #1 #2 #3
4438 {
4439   \l_@@_adjust_to_submatrix:nn { #1 } { #2 }
4440   \cs_if_free:cT { \l_@@_dotted_#1 - #2 }
4441   {
4442     \l_@@_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.

```

4443   \group_begin:
4444     \l_@@_open_shorten:
4445     \int_if_zero:nTF { #2 }
4446     { \color { nicematrix-first-col } }
4447     {
4448       \int_compare:nNnT { #2 } = \l_@@_last_col_int
4449       { \color { nicematrix-last-col } }
4450     }
4451     \keys_set:nn { nicematrix / xdots } { #3 }
4452     \l_@@_color:o \l_@@_xdots_color_tl
4453     \l_@@_actually_draw_Vdots:
4454   \group_end:
4455 }
4456 }

```

The command `\l_@@_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`.

The following function is also used by `\Vdotsfor`.

```

4457 \cs_new_protected:Npn \l_@@_actually_draw_Vdots:
4458 {

```

First, the case of a dotted line open on both sides.

```
4459     \bool_lazy_and:nTF \l_@@_initial_open_bool \l_@@_final_open_bool
```

We have to determine the  $x$ -value of the vertical rule that we will have to draw.

```
4460     {
4461     \@@_open_y_initial_dim:
4462     \@@_open_y_final_dim:
4463     \int_if_zero:nTF \l_@@_initial_j_int
```

We have a dotted line open on both sides in the “first column”.

```
4464     {
4465     \@@_qpoint:n { col - 1 }
4466     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4467     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4468     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4469     \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4470     }
4471     {
4472     \bool_lazy_and:nTF
4473     { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4474     { \int_compare_p:nNn \l_@@_initial_j_int = \g_@@_col_total_int }
```

We have a dotted line open on both sides in the “last column”.

```
4475     {
4476     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4477     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4478     \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4479     \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4480     \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4481     }
```

We have a dotted line open on both sides which is *not* in an exterior column.

```
4482     {
4483     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4484     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4485     \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4486     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4487     }
4488     }
4489 }
```

Now, the dotted line is *not* open on both sides (maybe open on only one side).

The boolean `\l_tmpa_bool` will indicate whether the column is of type 1 or may be considered as if.

```
4490     {
4491     \bool_set_false:N \l_tmpa_bool
4492     \bool_if:NF \l_@@_initial_open_bool
4493     {
4494     \bool_if:NF \l_@@_final_open_bool
4495     {
4496     \@@_set_initial_coords_from_anchor:n { south-west }
4497     \@@_set_final_coords_from_anchor:n { north-west }
4498     \bool_set:Nn \l_tmpa_bool
4499     { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4500     }
4501     }
```

Now, we try to determine whether the column is of type c or may be considered as if.

```
4502     \bool_if:NTF \l_@@_initial_open_bool
4503     {
4504     \@@_open_y_initial_dim:
4505     \@@_set_final_coords_from_anchor:n { north }
4506     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4507     }
4508     {
4509     \@@_set_initial_coords_from_anchor:n { south }
4510     \bool_if:NTF \l_@@_final_open_bool
```

```
4511         \l_@@_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.

```
4512         {
4513         \l_@@_set_final_coords_from_anchor:n { north }
4514         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4515         {
4516         \dim_set:Nn \l_@@_x_initial_dim
4517         {
4518         \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4519         \l_@@_x_initial_dim \l_@@_x_final_dim
4520         }
4521         }
4522         }
4523     }
4524 }
4525 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4526 \l_@@_draw_line:
4527 }
```

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.

```
4528 \cs_new_protected:Npn \l_@@_draw_Ddots:nmn #1 #2 #3
4529 {
4530     \l_@@_adjust_to_submatrix:nn { #1 } { #2 }
4531     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4532     {
4533         \l_@@_find_extremities_of_line:nmmn { #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.

```
4534     \group_begin:
4535     \l_@@_open_shorten:
4536     \keys_set:nn { nicematrix / xdots } { #3 }
4537     \l_@@_color:o \l_@@_xdots_color_tl
4538     \l_@@_actually_draw_Ddots:
4539     \group_end:
4540 }
4541 }
```

The command `\l_@@_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`.

```
4542 \cs_new_protected:Npn \l_@@_actually_draw_Ddots:
4543 {
4544     \bool_if:NTF \l_@@_initial_open_bool
4545     {
4546         \l_@@_open_y_initial_dim:
4547         \l_@@_open_x_initial_dim:
```

```

4548     }
4549     { \@@_set_initial_coords_from_anchor:n { south-east } }
4550 \bool_if:NTF \l_@@_final_open_bool
4551     {
4552     \@@_open_x_final_dim:
4553     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4554     }
4555     { \@@_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.

```

4556     \bool_if:NT \l_@@_parallelize_diags_bool
4557     {
4558     \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).

```

4559     \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  $\Delta_x$  and the  $\Delta_y$  of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```

4560     {
4561     \dim_gset:Nn \g_@@_delta_x_one_dim
4562     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4563     \dim_gset:Nn \g_@@_delta_y_one_dim
4564     { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4565     }

```

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`.

```

4566     {
4567     \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4568     {
4569     \dim_set:Nn \l_@@_y_final_dim
4570     {
4571     \l_@@_y_initial_dim +
4572     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4573     \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4574     }
4575     }
4576     }
4577 }
4578 \@@_draw_line:
4579 }

```

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.

```

4580 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4581 {
4582 \@@_adjust_to_submatrix:nn { #1 } { #2 }
4583 \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4584 {
4585 \@@_find_extremities_of_line:nmmm { #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.

```

4586 \group_begin:
4587 \@@_open_shorten:
4588 \keys_set:nn { nicematrix / xdots } { #3 }
4589 \@@_color:o \l_@@_xdots_color_tl
4590 \@@_actually_draw_Iddots:
4591 \group_end:
4592 }
4593 }

```



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.`

```

4594 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4595   {
4596     \bool_if:NTF \l_@@_initial_open_bool
4597       {
4598         \@@_open_y_initial_dim:
4599         \@@_open_x_initial_dim:
4600       }
4601     { \@@_set_initial_coords_from_anchor:n { south-west } }
4602   \bool_if:NTF \l_@@_final_open_bool
4603     {
4604       \@@_open_y_final_dim:
4605       \@@_open_x_final_dim:
4606     }
4607   { \@@_set_final_coords_from_anchor:n { north-east } }
4608   \bool_if:NT \l_@@_parallelize_diags_bool
4609     {
4610       \int_gincr:N \g_@@_iddots_int
4611       \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4612         {
4613           \dim_gset:Nn \g_@@_delta_x_two_dim
4614             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4615           \dim_gset:Nn \g_@@_delta_y_two_dim
4616             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4617         }
4618         {
4619           \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4620             {
4621               \dim_set:Nn \l_@@_y_final_dim
4622                 {
4623                   \l_@@_y_initial_dim +
4624                   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4625                   \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4626                 }
4627             }
4628         }
4629     }
4630   \@@_draw_line:
4631 }

```

## 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`

```

4632 \cs_new_protected:Npn \@@_draw_line:
4633 {
4634   \pgfrememberpicturepositiononpagetrue
4635   \pgf@relevantforpicturesizefalse
4636   \bool_lazy_or:nnTF
4637     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4638     \l_@@_dotted_bool
4639     \@@_draw_standard_dotted_line:
4640     \@@_draw_unstandard_dotted_line:
4641 }

```

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.

```

4642 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4643 {
4644   \begin { scope }
4645     \@@_draw_unstandard_dotted_line:o
4646     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4647 }

```

We have used the fact that, in PGF, un 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.

```

4648 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4649 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4650 {
4651   \@@_draw_unstandard_dotted_line:nooo
4652   { #1 }
4653   \l_@@_xdots_up_tl
4654   \l_@@_xdots_down_tl
4655   \l_@@_xdots_middle_tl
4656 }

```

The following Tikz styles are for the three labels (set by the symbols `_`, `^` and `=`) of a continuous line with a non-standard style.

```

4657 \hook_gput_code:nnn { begindocument } { . }
4658 {
4659   \IfPackageLoadedT { tikz }
4660   {
4661     \tikzset
4662     {
4663       @@_node_above / .style = { sloped , above } ,
4664       @@_node_below / .style = { sloped , below } ,
4665       @@_node_middle / .style =
4666       {
4667         sloped ,
4668         inner~sep = \c_@@_innersep_middle_dim
4669       }
4670     }
4671   }
4672 }

```

```

4673 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nmmn { n o o o }
4674 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nmmn #1 #2 #3 #4
4675 {

```

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  $\ell$  of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

4676 \dim_zero_new:N \l_@@_l_dim
4677 \dim_set:Nn \l_@@_l_dim
4678 {
4679 \fp_to_dim:n
4680 {
4681 sqrt
4682 (
4683 (\l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4684 +
4685 (\l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4686 )
4687 }
4688 }

```

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.

```

4689 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4690 {
4691 \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4692 \@@_draw_unstandard_dotted_line_i:
4693 }

```

If the key `xdots/horizontal-labels` has been used.

```

4694 \bool_if:NT \l_@@_xdots_h_labels_bool
4695 {
4696 \tikzset
4697 {
4698 \@@_node_above / .style = { auto = left } ,
4699 \@@_node_below / .style = { auto = right } ,
4700 \@@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4701 }
4702 }
4703 \tl_if_empty:nF { #4 }
4704 { \tikzset { \@@_node_middle / .append-style = { fill = white } } }
4705 \draw
4706 [ #1 ]
4707 ( \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).

```

4708 -- node [ \@@_node_middle ] { $ \scriptstyle #4 $ }
4709 node [ \@@_node_below ] { $ \scriptstyle #3 $ }
4710 node [ \@@_node_above ] { $ \scriptstyle #2 $ }
4711 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4712 \end { scope }
4713 }
4714 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4715 {
4716 \dim_set:Nn \l_tmpa_dim
4717 {
4718 \l_@@_x_initial_dim
4719 + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )

```

```

4720     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4721   }
4722   \dim_set:Nn \l_tmpb_dim
4723   {
4724     \l_@@_y_initial_dim
4725     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4726     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4727   }
4728   \dim_set:Nn \l_@@_tmpc_dim
4729   {
4730     \l_@@_x_final_dim
4731     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4732     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4733   }
4734   \dim_set:Nn \l_@@_tmpd_dim
4735   {
4736     \l_@@_y_final_dim
4737     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4738     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4739   }
4740   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4741   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4742   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4743   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4744 }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```

4745 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4746 {
4747   \group_begin:

```

The dimension `\l_@@_l_dim` is the length  $\ell$  of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

4748   \dim_zero_new:N \l_@@_l_dim
4749   \dim_set:Nn \l_@@_l_dim
4750   {
4751     \fp_to_dim:n
4752     {
4753       sqrt
4754       (
4755         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4756         +
4757         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4758       )
4759     }
4760   }

```

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.

```

4761   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4762   {
4763     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4764     \@@_draw_standard_dotted_line_i:
4765   }
4766   \group_end:
4767   \bool_lazy_all:nF
4768   {
4769     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4770     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4771     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }

```

```

4772     }
4773     \l_@@_labels_standard_dotted_line:
4774 }
4775 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4776 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4777 {

```

The number of dots will be  $\l_1\text{tmpa\_int} + 1$ .

```

4778     \int_set:Nn \l_1tmpa_int
4779     {
4780         \dim_ratio:nn
4781         {
4782             \l_@@_l_dim
4783             - \l_@@_xdots_shorten_start_dim
4784             - \l_@@_xdots_shorten_end_dim
4785         }
4786         \l_@@_xdots_inter_dim
4787     }

```

The dimensions  $\l_1\text{tmpa\_dim}$  and  $\l_1\text{tmpb\_dim}$  are the coordinates of the vector between two dots in the dotted line.

```

4788     \dim_set:Nn \l_1tmpa_dim
4789     {
4790         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4791         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4792     }
4793     \dim_set:Nn \l_1tmpb_dim
4794     {
4795         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4796         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4797     }

```

In the loop over the dots, the dimensions  $\l_1\text{@@_x\_initial\_dim}$  and  $\l_1\text{@@_y\_initial\_dim}$  will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

4798     \dim_gadd:Nn \l_@@_x_initial_dim
4799     {
4800         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4801         \dim_ratio:nn
4802         {
4803             \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
4804             + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4805         }
4806         { 2 \l_@@_l_dim }
4807     }
4808     \dim_gadd:Nn \l_@@_y_initial_dim
4809     {
4810         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4811         \dim_ratio:nn
4812         {
4813             \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
4814             + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4815         }
4816         { 2 \l_@@_l_dim }
4817     }
4818     \pgf@relevantforpicturesizefalse
4819     \int_step_inline:nnn \c_zero_int \l_1tmpa_int
4820     {
4821         \pgfpathcircle
4822         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4823         { \l_@@_xdots_radius_dim }
4824         \dim_add:Nn \l_@@_x_initial_dim \l_1tmpa_dim
4825         \dim_add:Nn \l_@@_y_initial_dim \l_1tmpb_dim
4826     }

```

```

4827   \pgfusepathqfill
4828 }

4829 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4830 {
4831   \pgfscope
4832   \pgftransformshift
4833   {
4834     \pgfpointlineattime { 0.5 }
4835     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4836     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4837   }
4838   \fp_set:Nn \l_tmpa_fp
4839   {
4840     atand
4841     (
4842       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4843       \l_@@_x_final_dim - \l_@@_x_initial_dim
4844     )
4845   }
4846   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4847   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4848   \tl_if_empty:NF \l_@@_xdots_middle_tl
4849   {
4850     \begin { pgfscope }
4851     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4852     \pgfnode
4853     { rectangle }
4854     { center }
4855     {
4856       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4857       {
4858         \c_math_toggle_token
4859         \scriptstyle \l_@@_xdots_middle_tl
4860         \c_math_toggle_token
4861       }
4862     }
4863     { }
4864     {
4865       \pgfsetfillcolor { white }
4866       \pgfusepath { fill }
4867     }
4868     \end { pgfscope }
4869   }
4870   \tl_if_empty:NF \l_@@_xdots_up_tl
4871   {
4872     \pgfnode
4873     { rectangle }
4874     { south }
4875     {
4876       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4877       {
4878         \c_math_toggle_token
4879         \scriptstyle \l_@@_xdots_up_tl
4880         \c_math_toggle_token
4881       }
4882     }
4883     { }
4884     { \pgfusepath { } }
4885   }
4886   \tl_if_empty:NF \l_@@_xdots_down_tl
4887   {
4888     \pgfnode

```

```

4889     { rectangle }
4890     { north }
4891     {
4892       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4893       {
4894         \c_math_toggle_token
4895         \scriptstyle \l_@@_xdots_down_tl
4896         \c_math_toggle_token
4897       }
4898     }
4899     { }
4900     { \pgfusepath { } }
4901   }
4902 \endpgfscope
4903 }

```

## 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.

```

4904 \hook_gput_code:nnn { begindocument } { . }
4905 {
4906   \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4907   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4908   \cs_new_protected:Npn \@@_Ldots
4909     { \@@_collect_options:n { \@@_Ldots_i } }
4910   \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4911     {
4912       \int_if_zero:nTF \c@jCol
4913       { \@@_error:nn { in~first~col } \Ldots }
4914       {
4915         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4916         { \@@_error:nn { in~last~col } \Ldots }
4917         {
4918           \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4919           { #1 , down = #2 , up = #3 , middle = #4 }
4920         }
4921       }
4922       \bool_if:NF \l_@@_nullify_dots_bool
4923       { \phantom { \ensuremath { \@@_old_ldots } } }
4924       \bool_gset_true:N \g_@@_empty_cell_bool
4925     }

4926   \cs_new_protected:Npn \@@_Cdots
4927     { \@@_collect_options:n { \@@_Cdots_i } }
4928   \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4929     {
4930       \int_if_zero:nTF \c@jCol
4931       { \@@_error:nn { in~first~col } \Cdots }
4932       {
4933         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int

```

```

4934         { \@@_error:nn { in~last~col } \Cdots }
4935         {
4936             \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4937             { #1 , down = #2 , up = #3 , middle = #4 }
4938         }
4939     }
4940     \bool_if:NF \l_@@_nullify_dots_bool
4941     { \phantom { \ensuremath { \@@_old_cdots } } }
4942     \bool_gset_true:N \g_@@_empty_cell_bool
4943 }

4944 \cs_new_protected:Npn \@@_Vdots
4945 { \@@_collect_options:n { \@@_Vdots_i } }
4946 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4947 {
4948     \int_if_zero:nTF \c@iRow
4949     { \@@_error:nn { in~first~row } \Vdots }
4950     {
4951         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4952         { \@@_error:nn { in~last~row } \Vdots }
4953         {
4954             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4955             { #1 , down = #2 , up = #3 , middle = #4 }
4956         }
4957     }
4958     \bool_if:NF \l_@@_nullify_dots_bool
4959     { \phantom { \ensuremath { \@@_old_vdots } } }
4960     \bool_gset_true:N \g_@@_empty_cell_bool
4961 }

4962 \cs_new_protected:Npn \@@_Ddots
4963 { \@@_collect_options:n { \@@_Ddots_i } }
4964 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4965 {
4966     \int_case:nnF \c@iRow
4967     {
4968         0 { \@@_error:nn { in~first~row } \Ddots }
4969         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4970     }
4971     {
4972         \int_case:nnF \c@jCol
4973         {
4974             0 { \@@_error:nn { in~first~col } \Ddots }
4975             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4976         }
4977         {
4978             \keys_set_known:nn { nicematrix / Ddots } { #1 }
4979             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4980             { #1 , down = #2 , up = #3 , middle = #4 }
4981         }
4982     }
4983 }
4984 \bool_if:NF \l_@@_nullify_dots_bool
4985 { \phantom { \ensuremath { \@@_old_ddots } } }
4986 \bool_gset_true:N \g_@@_empty_cell_bool
4987 }

4988 \cs_new_protected:Npn \@@_Iddots
4989 { \@@_collect_options:n { \@@_Iddots_i } }
4990 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
4991 {

```



```

4992     \int_case:nnF \c@iRow
4993     {
4994         0          { \@@_error:nn { in~first~row } \Iddots }
4995         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4996     }
4997     {
4998         \int_case:nnF \c@jCol
4999         {
5000             0          { \@@_error:nn { in~first~col } \Iddots }
5001             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5002         }
5003         {
5004             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5005             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5006             { #1 , down = #2 , up = #3 , middle = #4 }
5007         }
5008     }
5009     \bool_if:NF \l_@@_nullify_dots_bool
5010     { \phantom { \ensuremath { \@@_old_iddots } } }
5011     \bool_gset_true:N \g_@@_empty_cell_bool
5012 }
5013 }

```

End of the `\AddToHook`.

Despite its name, the following set of keys will be used for `\Ddots` but also for `\Iddots`.

```

5014 \keys_define:nn { nicematrix / Ddots }
5015 {
5016     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5017     draw-first .default:n = true ,
5018     draw-first .value_forbidden:n = true
5019 }

```

The command `\@@_Hspace:` will be linked to `\hspace` in `{NiceArray}`.

```

5020 \cs_new_protected:Npn \@@_Hspace:
5021 {
5022     \bool_gset_true:N \g_@@_empty_cell_bool
5023     \hspace
5024 }

```

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`.

```

5025 \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`.

```

5026 \cs_new:Npn \@@_Hdotsfor:
5027 {
5028     \bool_lazy_and:nnTF
5029     { \int_if_zero_p:n \c@jCol }
5030     { \int_if_zero_p:n \l_@@_first_col_int }
5031     {
5032         \bool_if:NTF \g_@@_after_col_zero_bool
5033         {
5034             \multicolumn { 1 } { c } { }
5035             \@@_Hdotsfor_i
5036         }
5037         { \@@_fatal:n { Hdotsfor~in~col~0 } }
5038     }
5039 }

```

```

5040     \multicolumn { 1 } { c } { }
5041     \@@_Hdotsfor_i
5042   }
5043 }

```

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:`).

```

5044 \hook_gput_code:nnn { begindocument } { . }
5045 {
5046   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5047   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

We don't put `!` before the last optional argument for homogeneity with `\Cdots`, etc. which have only one optional argument.

```

5048   \cs_new_protected:Npn \@@_Hdotsfor_i
5049     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5050   \exp_args:NNNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5051     {
5052     \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5053       {
5054         \@@_Hdotsfor:nnnn
5055         { \int_use:N \c@iRow }
5056         { \int_use:N \c@jCol }
5057         { #2 }
5058         {
5059           #1 , #3 ,
5060           down = \exp_not:n { #4 } ,
5061           up = \exp_not:n { #5 } ,
5062           middle = \exp_not:n { #6 }
5063         }
5064       }
5065     \prg_replicate:nn { #2 - 1 }
5066     {
5067       &
5068       \multicolumn { 1 } { c } { }
5069       \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5070     }
5071   }
5072 }

```

```

5073 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5074 {
5075   \bool_set_false:N \l_@@_initial_open_bool
5076   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5077   \int_set:Nn \l_@@_initial_i_int { #1 }
5078   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

5079   \int_compare:nNnTF { #2 } = \c_one_int
5080     {
5081     \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5082     \bool_set_true:N \l_@@_initial_open_bool
5083     }
5084     {
5085     \cs_if_exist:cTF
5086       {
5087         pgf @ sh @ ns @ \@@_env:
5088         - \int_use:N \l_@@_initial_i_int
5089         - \int_eval:n { #2 - 1 }
5090       }
5091     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5092     {

```

```

5093         \int_set:Nn \l_@@_initial_j_int { #2 }
5094         \bool_set_true:N \l_@@_initial_open_bool
5095     }
5096 }
5097 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5098 {
5099     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5100     \bool_set_true:N \l_@@_final_open_bool
5101 }
5102 {
5103     \cs_if_exist:cTF
5104     {
5105         pgf @ sh @ ns @ \@@_env:
5106         - \int_use:N \l_@@_final_i_int
5107         - \int_eval:n { #2 + #3 }
5108     }
5109     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5110     {
5111         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5112         \bool_set_true:N \l_@@_final_open_bool
5113     }
5114 }
5115 \group_begin:
5116 \@@_open_shorten:
5117 \int_if_zero:nTF { #1 }
5118 { \color { nicematrix-first-row } }
5119 {
5120     \int_compare:nNnT { #1 } = \g_@@_row_total_int
5121     { \color { nicematrix-last-row } }
5122 }
5123
5124 \keys_set:nn { nicematrix / xdots } { #4 }
5125 \@@_color:o \l_@@_xdots_color_tl
5126 \@@_actually_draw_Ldots:
5127 \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).

```

5128     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5129     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5130 }

5131 \hook_gput_code:nnn { begindocument } { . }
5132 {
5133     \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5134     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5135     \cs_new_protected:Npn \@@_Vdotsfor:
5136     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5137     \exp_args:NNNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5138     {
5139         \bool_gset_true:N \g_@@_empty_cell_bool
5140         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5141         {
5142             \@@_Vdotsfor:nnnn
5143             { \int_use:N \c@iRow }
5144             { \int_use:N \c@jCol }
5145             { #2 }
5146             {
5147                 #1 , #3 ,
5148                 down = \exp_not:n { #4 } ,
5149                 up = \exp_not:n { #5 } ,

```

```

5150         middle = \exp_not:n { #6 }
5151     }
5152 }
5153 }
5154 }

```

```

5155 \cs_new_protected:Npn \l_@@_Vdotsfor:nmmn #1 #2 #3 #4
5156 {
5157     \bool_set_false:N \l_@@_initial_open_bool
5158     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5159     \int_set:Nn \l_@@_initial_j_int { #2 }
5160     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it's a bit more complicated.

```

5161     \int_compare:nNnTF { #1 } = \c_one_int
5162     {
5163         \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5164         \bool_set_true:N \l_@@_initial_open_bool
5165     }
5166     {
5167         \cs_if_exist:cTF
5168         {
5169             pgf @ sh @ ns @ \l_@@_env:
5170             - \int_eval:n { #1 - 1 }
5171             - \int_use:N \l_@@_initial_j_int
5172         }
5173         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5174         {
5175             \int_set:Nn \l_@@_initial_i_int { #1 }
5176             \bool_set_true:N \l_@@_initial_open_bool
5177         }
5178     }
5179     \int_compare:nNnTF { #1 + #3 - 1 } = \c_iRow
5180     {
5181         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5182         \bool_set_true:N \l_@@_final_open_bool
5183     }
5184     {
5185         \cs_if_exist:cTF
5186         {
5187             pgf @ sh @ ns @ \l_@@_env:
5188             - \int_eval:n { #1 + #3 }
5189             - \int_use:N \l_@@_final_j_int
5190         }
5191         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5192         {
5193             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5194             \bool_set_true:N \l_@@_final_open_bool
5195         }
5196     }
5197     \group_begin:
5198     \l_@@_open_shorten:
5199     \int_if_zero:nTF { #2 }
5200     { \color { nicematrix-first-col } }
5201     {
5202         \int_compare:nNnT { #2 } = \g_@@_col_total_int
5203         { \color { nicematrix-last-col } }
5204     }
5205     \keys_set:nn { nicematrix / xdots } { #4 }
5206     \l_@@_color:o \l_@@_xdots_color_tl
5207     \l_@@_actually_draw_Vdots:
5208     \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:nnn`). This declaration is done by defining a special control sequence (to nil).

```

5209   \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5210       { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5211   }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5212 \NewDocumentCommand \@@_rotate: { 0 { } }
5213 {
5214   \peek_remove_spaces:n
5215     {
5216       \bool_gset_true:N \g_@@_rotate_bool
5217       \keys_set:nn { nicematrix / rotate } { #1 }
5218     }
5219 }

5220 \keys_define:nn { nicematrix / rotate }
5221 {
5222   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5223   c .value_forbidden:n = true ,
5224   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5225 }

```

## 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).<sup>13</sup>

```

5226 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5227 {
5228   \tl_if_empty:nTF { #2 }
5229     { #1 }
5230     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5231 }
5232 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5233 { \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).

```

5234 \hook_gput_code:nnn { begindocument } { . }
5235 {

```

---

<sup>13</sup>Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

5236 \cs_set_nopar:Npn \l_@@_argspec_tl
5237   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5238 \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5239 \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5240   {
5241     \group_begin:
5242     \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5243     \@@_color:o \l_@@_xdots_color_tl
5244     \use:e
5245     {
5246       \@@_line_i:nn
5247       { \@@_double_int_eval:n #2 - \q_stop }
5248       { \@@_double_int_eval:n #3 - \q_stop }
5249     }
5250     \group_end:
5251   }
5252 }

5253 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5254   {
5255     \bool_set_false:N \l_@@_initial_open_bool
5256     \bool_set_false:N \l_@@_final_open_bool
5257     \bool_lazy_or:nnTF
5258     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5259     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5260     { \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 } }

```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5261   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5262 }

5263 \hook_gput_code:nnn { begindocument } { . }
5264 {
5265   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5266   {

```

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:.`

```

5267     \c_@@_pgfortikzpicture_tl
5268     \@@_draw_line_iii:nn { #1 } { #2 }
5269     \c_@@_endpgfortikzpicture_tl
5270   }
5271 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5272 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5273   {
5274     \pgfrememberpicturepositiononpagetrue
5275     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5276     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5277     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5278     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5279     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5280     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5281     \@@_draw_line:
5282   }

```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don’t use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

## 20 The command `\RowStyle`

`\g_@@_row_style_tl` may contain several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

Then, `\g_@@_row_style_tl` will be inserted in all the cells of the array (and also in both components of a `\diagbox` in a cell of in a mono-row block).

The test `\@@_if_row_less_then:nn` ensures that the instructions are inserted only if you are in a row which is (still) in the scope of that instructions (which depends on the value of the key `nb-rows` of `\RowStyle`).

That test will be active even in an expandable context because `\@@_if_row_less_then:nn` is *not* protected.

`#1` is the first row *after* the scope of the instructions in `#2`

```
5283 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5284   { \int_compare:nNnT { \c@iRow } < { #1 } { #2 } }
```

`\@@_put_in_row_style` will be used several times by `\RowStyle`.

```
5285 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
5286 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5287   {
5288     \tl_gput_right:Ne \g_@@_row_style_tl
5289     {
```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can't be replaced by a protected version of `\@@_if_row_less_than:nn`.

```
5290     \exp_not:N
5291     \@@_if_row_less_than:nn
5292     { \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`).

```
5293     { \exp_not:n { #1 } \scan_stop: }
5294   }
5295 }
```

```
5296 \keys_define:nn { nicematrix / RowStyle }
5297   {
5298     cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5299     cell-space-top-limit .value_required:n = true ,
5300     cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5301     cell-space-bottom-limit .value_required:n = true ,
5302     cell-space-limits .meta:n =
5303     {
5304       cell-space-top-limit = #1 ,
5305       cell-space-bottom-limit = #1 ,
5306     } ,
5307     color .tl_set:N = \l_@@_color_tl ,
5308     color .value_required:n = true ,
5309     bold .bool_set:N = \l_@@_bold_row_style_bool ,
5310     bold .default:n = true ,
5311     nb-rows .code:n =
5312     \str_if_eq:eeTF { #1 } { * }
5313     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5314     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5315     nb-rows .value_required:n = true ,
5316     rowcolor .tl_set:N = \l_tmpa_tl ,
5317     rowcolor .value_required:n = true ,
5318     unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5319   }
```

```

5320 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5321 {
5322   \group_begin:
5323   \tl_clear:N \l_tmpa_tl
5324   \tl_clear:N \l_@@_color_tl
5325   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5326   \dim_zero:N \l_tmpa_dim
5327   \dim_zero:N \l_tmpb_dim
5328   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

5329   \tl_if_empty:NF \l_tmpa_tl
5330   {

```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row).

```

5331     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5332     {

```

The command `\@@_exp_color_arg:No` is *fully expandable*.

```

5333         \@@_exp_color_arg:No \@@_rectanglecolor \l_tmpa_tl
5334         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5335         { \int_use:N \c@iRow - * }
5336     }

```

Then, the other rows (if there is several rows).

```

5337     \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5338     {
5339       \tl_gput_right:Ne \g_@@_pre_code_before_tl
5340       {
5341         \@@_exp_color_arg:No \@@_rowcolor \l_tmpa_tl
5342         {
5343           \int_eval:n { \c@iRow + 1 }
5344           - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5345         }
5346       }
5347     }
5348   }
5349   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

5350   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5351   {
5352     \@@_put_in_row_style:e
5353     {
5354       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5355       {

```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```

5356         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5357         { \dim_use:N \l_tmpa_dim }
5358       }
5359     }
5360   }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

5361   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5362   {
5363     \@@_put_in_row_style:e
5364     {
5365       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5366       {
5367         \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5368         { \dim_use:N \l_tmpb_dim }
5369       }
5370     }
5371   }

```



`\l_@@_color_tl` is the value of the key color of `\RowStyle`.

```

5372   \tl_if_empty:NF \l_@@_color_tl
5373   {
5374     \@@_put_in_row_style:e
5375     {
5376       \mode_leave_vertical:
5377       \@@_color:n { \l_@@_color_tl }
5378     }
5379   }

```

`\l_@@_bold_row_style_bool` is the value of the key bold.

```

5380   \bool_if:NT \l_@@_bold_row_style_bool
5381   {
5382     \@@_put_in_row_style:n
5383     {
5384       \exp_not:n
5385       {
5386         \if_mode_math:
5387         \c_math_toggle_token
5388         \bfseries \boldmath
5389         \c_math_toggle_token
5390         \else:
5391         \bfseries \boldmath
5392         \fi:
5393       }
5394     }
5395   }
5396   \group_end:
5397   \g_@@_row_style_tl
5398   \ignorespaces
5399 }

```

## 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).

```

5400 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5401 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
5402 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5403 {

```

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.

```
5404 \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`.

```
5405 \str_if_in:nnF { #1 } { !! }
5406 {
5407 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```
5408 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5409 }
5410 \int_if_zero:nTF \l_tmpa_int
```

First, the case where the color is a *new* color (not in the sequence).

```
5411 {
5412 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5413 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5414 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5415 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5416 }
```

The following command must be used within a `\pgfpicture`.

```
5417 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5418 {
5419 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5420 {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```
5421 \group_begin:
5422 \pgfsetcornersarced
5423 {
5424 \pgfpoint
5425 { \l_@@_tab_rounded_corners_dim }
5426 { \l_@@_tab_rounded_corners_dim }
5427 }
```

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.

```
5428 \bool_if:NTF \l_@@_hvlines_bool
5429 {
5430 \pgfpathrectanglecorners
5431 {
5432 \pgfpointadd
5433 { \@@_qpoint:n { row-1 } }
5434 { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5435 }
5436 {
5437 \pgfpointadd
5438 {
5439 \@@_qpoint:n
5440 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5441 }
5442 { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5443 }
5444 }
5445 {
```

```

5446     \pgfpathrectanglecorners
5447     { \@@_qpoint:n { row-1 } }
5448     {
5449     \pgfpointadd
5450     {
5451     \@@_qpoint:n
5452     { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5453     }
5454     { \pgfpoint \c_zero_dim \arrayrulewidth }
5455     }
5456     }
5457     \pgfusepath { clip }
5458     \group_end:

```

The TeX group was for \pgfsetcornersarced.

```

5459     }
5460 }

```

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).

```

5461 \cs_new_protected:Npn \@@_actually_color:
5462 {
5463 \pgfpicture
5464 \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.

```

5465 \@@_clip_with_rounded_corners:
5466 \seq_map_indexed_inline:Nn \g_@@_colors_seq
5467 {
5468 \int_compare:nNnTF { ##1 } = \c_one_int
5469 {
5470 \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5471 \use:c { g_@@_color _ 1 _tl }
5472 \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5473 }
5474 {
5475 \begin { pgfscope }
5476 \@@_color_opacity ##2
5477 \use:c { g_@@_color _ ##1 _tl }
5478 \tl_gclear:c { g_@@_color _ ##1 _tl }
5479 \pgfusepath { fill }
5480 \end { pgfscope }
5481 }
5482 }
5483 \endpgfpicture
5484 }

```

The following command will extract the potential key opacity in its optional argument (between square brackets) and (of course) then apply the command \color.

```

5485 \cs_new_protected:Npn \@@_color_opacity
5486 {
5487 \peek_meaning:NTF [
5488 { \@@_color_opacity:w }
5489 { \@@_color_opacity:w [ ] }
5490 }

```

The command \@@\_color\_opacity:w takes in as argument only the optional argument. One may consider that the second argument (the actual definition of the color) is provided by curryfication.

```

5491 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5492 {
5493 \tl_clear:N \l_tmpa_tl
5494 \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.

```

5495     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5496     \tl_if_empty:NTF \l_tmpb_tl
5497         { \@declaredcolor }
5498         { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5499     }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5500 \keys_define:nn { nicematrix / color-opacity }
5501 {
5502     opacity .tl_set:N          = \l_tmpa_tl ,
5503     opacity .value_required:n = true
5504 }

5505 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5506 {
5507     \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5508     \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5509     \@@_cartesian_path:
5510 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5511 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5512 {
5513     \tl_if_blank:nF { #2 }
5514     {
5515         \@@_add_to_colors_seq:en
5516         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5517         { \@@_cartesian_color:nn { #3 } { - } }
5518     }
5519 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5520 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5521 {
5522     \tl_if_blank:nF { #2 }
5523     {
5524         \@@_add_to_colors_seq:en
5525         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5526         { \@@_cartesian_color:nn { - } { #3 } }
5527     }
5528 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5529 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5530 {
5531     \tl_if_blank:nF { #2 }
5532     {
5533         \@@_add_to_colors_seq:en
5534         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5535         { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5536     }
5537 }

```

The last argument is the radius of the corners of the rectangle.

```

5538 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5539 {
5540     \tl_if_blank:nF { #2 }

```

```

5541     {
5542     \@@_add_to_colors_seq:en
5543     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5544     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5545     }
5546 }

```

The last argument is the radius of the corners of the rectangle.

```

5547 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5548 {
5549   \@@_cut_on_hyphen:w #1 \q_stop
5550   \tl_clear_new:N \l_@@_tmpc_tl
5551   \tl_clear_new:N \l_@@_tmpd_tl
5552   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5553   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5554   \@@_cut_on_hyphen:w #2 \q_stop
5555   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5556   \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`.

```

5557   \@@_cartesian_path:n { #3 }
5558 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5559 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5560 {
5561   \clist_map_inline:nn { #3 }
5562   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5563 }

5564 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5565 {
5566   \int_step_inline:nn \c@iRow
5567   {
5568     \int_step_inline:nn \c@jCol
5569     {
5570       \int_if_even:nTF { #####1 + ##1 }
5571       { \@@_cellcolor [ #1 ] { #2 } }
5572       { \@@_cellcolor [ #1 ] { #3 } }
5573       { ##1 - #####1 }
5574     }
5575   }
5576 }

```

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”.

```

5577 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5578 {
5579   \@@_rectanglecolor [ #1 ] { #2 }
5580   { 1 - 1 }
5581   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5582 }

5583 \keys_define:nn { nicematrix / rowcolors }
5584 {
5585   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5586   respect-blocks .default:n = true ,
5587   cols .tl_set:N = \l_@@_cols_tl ,

```

```

5588     restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5589     restart .default:n = true ,
5590     unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5591 }

```

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`.

```

5592 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5593 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5594     \group_begin:
5595     \seq_clear_new:N \l_@@_colors_seq
5596     \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5597     \tl_clear_new:N \l_@@_cols_tl
5598     \cs_set_nopar:Npn \l_@@_cols_tl { - }
5599     \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).

```

5600     \int_zero_new:N \l_@@_color_int
5601     \int_set_eq:NN \l_@@_color_int \c_one_int
5602     \bool_if:NT \l_@@_respect_blocks_bool
5603     {

```

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`).

```

5604         \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5605         \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5606         { \@@_not_in_exterior_p:nnnnn ##1 }
5607     }
5608     \pgfpicture
5609     \pgf@relevantforpicturesizefalse

```

**#2** is the list of intervals of rows.

```

5610     \clist_map_inline:nn { #2 }
5611     {
5612         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5613         \tl_if_in:NnTF \l_tmpa_tl { - }
5614         { \@@_cut_on_hyphen:w ##1 \q_stop }
5615         { \tl_set:No \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.

```

5616         \int_set:Nn \l_tmpa_int \l_tmpa_tl
5617         \int_set:Nn \l_@@_color_int
5618         { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5619         \int_zero_new:N \l_@@_tmpc_int
5620         \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5621         \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5622         {

```

We will compute in `\l_tmpb_int` the last row of the "block".

```

5623             \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).

```

5624         \bool_if:NT \l_@@_respect_blocks_bool
5625         {
5626             \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5627             { \@@_intersect_our_row_p:nnnnn #####1 }
5628             \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5629     }
5630     \tl_set:No \l_@@_rows_tl
5631     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
\l_@@_tmpc_tl will be the color that we will use.
5632     \tl_clear_new:N \l_@@_color_tl
5633     \tl_set:Ne \l_@@_color_tl
5634     {
5635         \@@_color_index:n
5636         {
5637             \int_mod:nn
5638             { \l_@@_color_int - 1 }
5639             { \seq_count:N \l_@@_colors_seq }
5640             + 1
5641         }
5642     }
5643     \tl_if_empty:NF \l_@@_color_tl
5644     {
5645         \@@_add_to_colors_seq:ee
5646         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5647         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5648     }
5649     \int_incr:N \l_@@_color_int
5650     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5651 }
5652 }
5653 \endpgfpicture
5654 \group_end:
5655 }

```

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.

```

5656 \cs_new:Npn \@@_color_index:n #1
5657 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5658     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5659     { \@@_color_index:n { #1 - 1 } }
5660     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5661 }

```

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.

```

5662 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5663 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around `#3` and `#4` are mandatory.

```

5664 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5665 {
5666     \int_compare:nNnT { #3 } > \l_tmpb_int
5667     { \int_set:Nn \l_tmpb_int { #3 } }
5668 }

```

```

5669 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5670 {
5671     \int_if_zero:nTF { #4 }
5672     \prg_return_false:
5673     {
5674         \int_compare:nNnTF { #2 } > \c@jCol

```

```

5675     \prg_return_false:
5676     \prg_return_true:
5677   }
5678 }

```

The following command return true when the block intersects the row \l\_tmpa\_int.

```

5679 \prg_new_conditional:Nnn \@@_intersect_our_row:nmmnn p
5680 {
5681   \int_compare:nNnTF { #1 } > \l_tmpa_int
5682     \prg_return_false:
5683     {
5684       \int_compare:nNnTF \l_tmpa_int > { #3 }
5685         \prg_return_false:
5686         \prg_return_true:
5687     }
5688 }

```

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).

```

5689 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5690 {
5691   \dim_compare:nNnTF { #1 } = \c_zero_dim
5692     {
5693       \bool_if:NTF
5694         \l_@@_nocolor_used_bool
5695         \@@_cartesian_path_normal_ii:
5696         {
5697           \clist_if_empty:NTF \l_@@_corners_cells_clist
5698             { \@@_cartesian_path_normal_i:n { #1 } }
5699             \@@_cartesian_path_normal_ii:
5700         }
5701     }
5702     { \@@_cartesian_path_normal_i:n { #1 } }
5703 }

```

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.

```

5704 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5705 {
5706   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5707   \clist_map_inline:Nn \l_@@_cols_tl
5708     {
5709       \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5710       \tl_if_in:NnTF \l_tmpa_tl { - }
5711         { \@@_cut_on_hyphen:w ##1 \q_stop }
5712         { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5713       \tl_if_empty:NTF \l_tmpa_tl
5714         { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5715         {
5716           \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5717             { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5718         }
5719       \tl_if_empty:NTF \l_tmpb_tl
5720         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5721         {

```



```

5722     \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5723     { \tl_set:No \l_tmpb_tl { \int_use:N \c_jCol } }
5724   }
5725   \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5726   { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5727     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5728     \@@_qpoint:n { col - \l_tmpa_tl }
5729     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5730     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5731     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5732     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5733     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5734   \clist_map_inline:Nn \l_@@_rows_tl
5735   {
5736     \cs_set_nopar:Npn \l_tmpa_tl { #####1 }
5737     \tl_if_in:NnTF \l_tmpa_tl { - }
5738     { \@@_cut_on_hyphen:w #####1 \q_stop }
5739     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5740     \tl_if_empty:NTF \l_tmpa_tl
5741     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5742     {
5743       \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5744       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5745     }
5746     \tl_if_empty:NTF \l_tmpb_tl
5747     { \tl_set:No \l_tmpb_tl { \int_use:N \c_iRow } }
5748     {
5749       \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5750       { \tl_set:No \l_tmpb_tl { \int_use:N \c_iRow } }
5751     }
5752     \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5753     { \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`.

```

5754     \cs_if_exist:cF
5755     { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5756     {
5757       \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5758       \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5759       \@@_qpoint:n { row - \l_tmpa_tl }
5760       \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5761       \pgfpathrectanglecorners
5762       { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5763       { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5764     }
5765   }
5766 }
5767 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5768 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5769 {
5770   \@@_expand_clist:NN \l_@@_cols_tl \c_jCol
5771   \@@_expand_clist:NN \l_@@_rows_tl \c_iRow

```

We begin the loop over the columns.

```

5772   \clist_map_inline:Nn \l_@@_cols_tl
5773   {
5774     \@@_qpoint:n { col - ##1 }
5775     \int_compare:nNnTF \l_@@_first_col_int = { ##1 }

```

```

5776     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5777     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5778     \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5779     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5780     \clist_map_inline:Nn \l_@@_rows_tl
5781     {
5782         \@@_if_in_corner:nF { #####1 - ##1 }
5783         {
5784             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5785             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5786             \@@_qpoint:n { row - #####1 }
5787             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5788             \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5789             {
5790                 \pgfpathrectanglecorners
5791                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5792                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5793             }
5794         }
5795     }
5796 }
5797 }

```

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`).

```

5798 \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

```

5799 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5800 {
5801     \bool_set_true:N \l_@@_nocolor_used_bool
5802     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5803     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5804     \clist_map_inline:Nn \l_@@_rows_tl
5805     {
5806         \clist_map_inline:Nn \l_@@_cols_tl
5807         { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
5808     }
5809 }

```

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.

```

5810 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5811 {
5812     \clist_set_eq:NN \l_tmpa_clist #1
5813     \clist_clear:N #1
5814     \clist_map_inline:Nn \l_tmpa_clist
5815     {
5816         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5817         \tl_if_in:NnTF \l_tmpa_tl { - }
5818         { \@@_cut_on_hyphen:w ##1 \q_stop }
5819         { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5820     }
5821     \bool_lazy_or:nnT
5822     { \str_if_eq_p:ee \l_tmpa_tl { * } }
5823     { \tl_if_blank_p:o \l_tmpa_tl }

```

```

5823     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5824 \bool_lazy_or:nnT
5825   { \str_if_eq_p:ee \l_tmpb_tl { * } }
5826   { \tl_if_blank_p:o \l_tmpb_tl }
5827   { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5828 \int_compare:nNnT \l_tmpb_tl > #2
5829   { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5830 \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5831   { \clist_put_right:Nn #1 { ###1 } }
5832 }
5833 }

```

When the user uses the key `color-inside`, the following command will be linked to `\cellcolor` in the tabular.

```

5834 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5835 {
5836   \@@_test_color_inside:
5837   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5838   {

```

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`).

```

5839     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5840     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5841   }
5842   \ignorespaces
5843 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolor` in the tabular.

```

5844 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5845 {
5846   \@@_test_color_inside:
5847   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5848   {
5849     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5850     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5851     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5852   }
5853   \ignorespaces
5854 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5855 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5856 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around `#2` and `#3` are mandatory.

When the user uses the key `color-inside`, the following command will be linked to `\rowlistcolors` in the tabular.

```

5857 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5858 {
5859   \@@_test_color_inside:
5860   \peek_remove_spaces:n
5861   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5862 }

5863 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5864 {

```

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`.

```

5865     \seq_gclear:N \g_tmpa_seq
5866     \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5867       { \@@_rowlistcolors_tabular_i:nmmn ##1 }
5868     \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`.

```

5869     \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5870     {
5871       { \int_use:N \c@iRow }
5872       { \exp_not:n { #1 } }
5873       { \exp_not:n { #2 } }
5874       { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5875     }
5876   }

```

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`).

```

5877 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nmmn #1 #2 #3 #4
5878 {
5879   \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).

```

5880     { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5881     {
5882       \tl_gput_right:Ne \g_@@_pre_code_before_tl
5883       {
5884         \@@_rowlistcolors
5885         [ \exp_not:n { #2 } ]
5886         { #1 - \int_eval:n { \c@iRow - 1 } }
5887         { \exp_not:n { #3 } }
5888         [ \exp_not:n { #4 } ]
5889       }
5890     }
5891   }

```

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.

```

5892 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5893 {
5894   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5895     { \@@_rowlistcolors_tabular_ii:nmmn ##1 }
5896   \seq_gclear:N \g_@@_rowlistcolors_seq
5897 }

```

```

5898 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nmmm #1 #2 #3 #4
5899 {
5900   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5901   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5902 }

```

The first mandatory argument of the command `\@@_rowlistcolors` which is written 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.

```

5903 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5904 {

```

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).

```

5905   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5906   {

```

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).

```

5907     \tl_gput_left:Ne \g_@@_pre_code_before_tl
5908     {
5909       \exp_not:N \columncolor [ #1 ]
5910       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5911     }
5912   }
5913 }

```

```

5914 \hook_gput_code:nnn { begindocument } { . }
5915 {
5916   \IfPackageLoadedTF { colortbl }
5917   {
5918     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5919     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5920     \cs_new_protected:Npn \@@_revert_colortbl:
5921     {
5922       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
5923       {
5924         \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5925         \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5926       }
5927     }
5928   }
5929   { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5930 }

```

## 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`.

```
5931 \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`.

```
5932 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5933 {
5934   \int_if_zero:nTF \l_@@_first_col_int
5935     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5936     {
5937       \int_if_zero:nTF \c@jCol
5938         {
5939           \int_compare:nNnF \c@iRow = { -1 }
5940             { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5941         }
5942         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5943     }
5944 }
```

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.

```
5945 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5946 {
5947   \int_if_zero:nF \c@iRow
5948   {
5949     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
5950     {
5951       \int_compare:nNnT \c@jCol > \c_zero_int
5952       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5953     }
5954   }
5955 }
```

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`).

## 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.

```
5956 \keys_define:nn { nicematrix / Rules }
5957 {
5958   position .int_set:N = \l_@@_position_int ,
5959   position .value_required:n = true ,
5960   start .int_set:N = \l_@@_start_int ,
5961   end .code:n =
5962     \bool_lazy_or:nnTF
5963       { \tl_if_empty_p:n { #1 } }
5964       { \str_if_eq_p:ee { #1 } { last } }
5965       { \int_set_eq:NN \l_@@_end_int \c@jCol }
5966       { \int_set:Nn \l_@@_end_int { #1 } }
5967 }
```

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.

```

5968 \keys_define:nn { nicematrix / RulesBis }
5969   {
5970     multiplicity .int_set:N = \l_@@_multiplicity_int ,
5971     multiplicity .initial:n = 1 ,
5972     dotted .bool_set:N = \l_@@_dotted_bool ,
5973     dotted .initial:n = false ,
5974     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]`.

```

5975   color .code:n =
5976     \@@_set_CT@arc@:n { #1 }
5977     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
5978   color .value_required:n = true ,
5979   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5980   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.

```

5981   tikz .code:n =
5982     \IfPackageLoadedTF { tikz }
5983       { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
5984       { \@@_error:n { tikz-without-tikz } } ,
5985   tikz .value_required:n = true ,
5986   total-width .dim_set:N = \l_@@_rule_width_dim ,
5987   total-width .value_required:n = true ,
5988   width .meta:n = { total-width = #1 } ,
5989   unknown .code:n = \@@_error:n { Unknow-key-for-RulesBis }
5990   }

```

## The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

5991 \cs_new_protected:Npn \@@_vline:n #1
5992   {

```

The group is for the options.

```

5993     \group_begin:
5994     \int_set_eq:NN \l_@@_end_int \c@iRow
5995     \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).

```

5996     \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
5997     \@@_vline_i:
5998     \group_end:
5999   }

```

```

6000 \cs_new_protected:Npn \@@_vline_i:
6001   {

```

`\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`.

```

6002     \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
6003     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6004     \l_tmpa_tl
6005     {

```

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.

```

6006     \bool_gset_true:N \g_tmpa_bool
6007     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6008       { \@@_test_vline_in_block:nnnnn ##1 }
6009     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6010       { \@@_test_vline_in_block:nnnnn ##1 }
6011     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6012       { \@@_test_vline_in_stroken_block:nnnn ##1 }
6013     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6014     \bool_if:NTF \g_tmpa_bool
6015       {
6016         \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.

```

6017         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6018       }
6019     {
6020       \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6021       {
6022         \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6023         \@@_vline_ii:
6024         \int_zero:N \l_@@_local_start_int
6025       }
6026     }
6027   }
6028 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6029 {
6030   \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6031   \@@_vline_ii:
6032 }
6033 }

```

```

6034 \cs_new_protected:Npn \@@_test_in_corner_v:
6035 {
6036   \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6037   {
6038     \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6039     { \bool_set_false:N \g_tmpa_bool }
6040   }
6041   {
6042     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6043     {
6044       \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6045       { \bool_set_false:N \g_tmpa_bool }
6046       {
6047         \@@_if_in_corner:nT
6048           { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6049           { \bool_set_false:N \g_tmpa_bool }
6050       }
6051     }
6052   }
6053 }

```

```

6054 \cs_new_protected:Npn \@@_vline_ii:
6055 {
6056   \tl_clear:N \l_@@_tikz_rule_tl
6057   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl

```



```

6058 \bool_if:NTF \l_@@_dotted_bool
6059   \@@_vline_iv:
6060   {
6061     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6062     \@@_vline_iii:
6063     \@@_vline_v:
6064   }
6065 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

6066 \cs_new_protected:Npn \@@_vline_iii:
6067 {
6068   \pgfpicture
6069   \pgfrememberpicturepositiononpagetrue
6070   \pgf@relevantforpicturesizefalse
6071   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6072   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6073   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6074   \dim_set:Nn \l_tmpb_dim
6075   {
6076     \pgf@x
6077     - 0.5 \l_@@_rule_width_dim
6078     +
6079     ( \arrayrulewidth * \l_@@_multiplicity_int
6080       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6081   }
6082   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6083   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6084   \bool_lazy_all:nT
6085   {
6086     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6087     { \cs_if_exist_p:N \CT@drsc@ }
6088     { ! \tl_if_blank_p:o \CT@drsc@ }
6089   }
6090   {
6091     \group_begin:
6092     \CT@drsc@
6093     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6094     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6095     \dim_set:Nn \l_@@_tmpd_dim
6096     {
6097       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6098       * ( \l_@@_multiplicity_int - 1 )
6099     }
6100     \pgfpathrectanglecorners
6101     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6102     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6103     \pgfusepath { fill }
6104     \group_end:
6105   }
6106   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6107   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6108   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6109   {
6110     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6111     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6112     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6113     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6114   }
6115   \CT@arc@
6116   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6117   \pgfsetrectcap
6118   \pgfusepathqstroke

```

```

6119   \endpgfpicture
6120 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6121 \cs_new_protected:Npn \@@_vline_iv:
6122 {
6123   \pgfpicture
6124   \pgfrememberpicturepositiononpagetrue
6125   \pgf@relevantforpicturesizefalse
6126   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6127   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6128   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6129   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6130   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6131   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6132   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6133   \CT@arc@
6134   \@@_draw_line:
6135   \endpgfpicture
6136 }

```

The following code is for the case when the user uses the key `tikz`.

```

6137 \cs_new_protected:Npn \@@_vline_v:
6138 {
6139   \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).

```

6140   \CT@arc@
6141   \tl_if_empty:NF \l_@@_rule_color_tl
6142   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6143   \pgfrememberpicturepositiononpagetrue
6144   \pgf@relevantforpicturesizefalse
6145   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6146   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6147   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6148   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6149   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6150   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6151   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6152   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6153   ( \l_tmpb_dim , \l_tmpa_dim ) --
6154   ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6155   \end {tikzpicture }
6156 }

```

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).

```

6157 \cs_new_protected:Npn \@@_draw_vlines:
6158 {
6159   \int_step_inline:nnn
6160   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6161   {
6162     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6163     \c@jCol
6164     { \int_eval:n { \c@jCol + 1 } }
6165   }
6166   {
6167     \tl_if_eq:NNF \l_@@_vlines_clist \c_@@_all_tl
6168     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6169     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }

```

```

6170     }
6171 }

```

## 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}`.

```

6172 \cs_new_protected:Npn \@@_hline:n #1
6173 {

```

The group is for the options.

```

6174     \group_begin:
6175     \int_zero_new:N \l_@@_end_int
6176     \int_set_eq:NN \l_@@_end_int \c@jCol
6177     \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6178     \@@_hline_i:
6179     \group_end:
6180 }

```

```

6181 \cs_new_protected:Npn \@@_hline_i:

```

```

6182 {
6183     \int_zero_new:N \l_@@_local_start_int
6184     \int_zero_new:N \l_@@_local_end_int

```

`\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`.

```

6185     \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6186     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6187         \l_tmpb_tl
6188     {

```

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.

```

6189         \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6190         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6191             { \@@_test_hline_in_block:nnnnn ##1 }
6192         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6193             { \@@_test_hline_in_block:nnnnn ##1 }
6194         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6195             { \@@_test_hline_in_stroken_block:nnnn ##1 }
6196         \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6197         \bool_if:NTF \g_tmpa_bool
6198             {
6199             \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.

```

6200             { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6201         }
6202     {
6203         \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6204         {
6205             \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6206             \@@_hline_ii:
6207             \int_zero:N \l_@@_local_start_int
6208         }
6209     }
6210 }
6211 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int

```

```

6212     {
6213         \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6214         \@@_hline_ii:
6215     }
6216 }

6217 \cs_new_protected:Npn \@@_test_in_corner_h:
6218 {
6219     \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6220     {
6221         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6222         { \bool_set_false:N \g_tmpa_bool }
6223     }
6224     {
6225         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6226         {
6227             \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6228             { \bool_set_false:N \g_tmpa_bool }
6229             {
6230                 \@@_if_in_corner:nT
6231                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6232                 { \bool_set_false:N \g_tmpa_bool }
6233             }
6234         }
6235     }
6236 }

6237 \cs_new_protected:Npn \@@_hline_ii:
6238 {
6239     \tl_clear:N \l_@@_tikz_rule_tl
6240     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6241     \bool_if:NTF \l_@@_dotted_bool
6242     \@@_hline_iv:
6243     {
6244         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6245         \@@_hline_iii:
6246         \@@_hline_v:
6247     }
6248 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6249 \cs_new_protected:Npn \@@_hline_iii:
6250 {
6251     \pgfpicture
6252     \pgfrememberpicturepositiononpagetrue
6253     \pgf@relevantforpicturesizefalse
6254     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6255     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6256     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6257     \dim_set:Nn \l_tmpb_dim
6258     {
6259         \pgf@y
6260         - 0.5 \l_@@_rule_width_dim
6261         +
6262         ( \arrayrulewidth * \l_@@_multiplicity_int
6263           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6264     }
6265     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6266     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6267     \bool_lazy_all:nT
6268     {

```

```

6269     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6270     { \cs_if_exist_p:N \CT@drsc@ }
6271     { ! \tl_if_blank_p:o \CT@drsc@ }
6272   }
6273   {
6274     \group_begin:
6275     \CT@drsc@
6276     \dim_set:Nn \l_@@_tmpd_dim
6277     {
6278       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6279       * ( \l_@@_multiplicity_int - 1 )
6280     }
6281     \pgfpathrectanglecorners
6282     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6283     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6284     \pgfusepathqfill
6285     \group_end:
6286   }
6287   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6288   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6289   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6290   {
6291     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6292     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6293     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6294     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6295   }
6296   \CT@arc@
6297   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6298   \pgfsetrectcap
6299   \pgfusepathqstroke
6300   \endpgfpicture
6301 }

```

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6302 \cs_new_protected:Npn \@@_hline_iv:
6303   {
6304     \pgfpicture
6305     \pgfrememberpicturepositiononpagetrue
6306     \pgf@relevantforpicturesizefalse
6307     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6308     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6309     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6310     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6311     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x

```

```

6312 \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6313 {
6314   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6315   \bool_if:NF \g_@@_delims_bool
6316   { \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.

```

6317   \tl_if_eq:NnF \g_@@_left_delim_tl (
6318     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6319   )
6320 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6321 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6322 \int_compare:nNnT \l_@@_local_end_int = \c@jCol
6323 {
6324   \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6325   \bool_if:NF \g_@@_delims_bool
6326   { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6327   \tl_if_eq:NnF \g_@@_right_delim_tl )
6328   { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6329 }
6330 \CT@arc@
6331 \@@_draw_line:
6332 \endpgfpicture
6333 }

```

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`).

```

6334 \cs_new_protected:Npn \@@_hline_v:
6335 {
6336   \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).

```

6337   \CT@arc@
6338   \tl_if_empty:NF \l_@@_rule_color_tl
6339   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6340   \pgfrememberpicturepositiononpagetrue
6341   \pgf@relevantforpicturesizefalse
6342   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6343   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6344   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6345   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6346   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6347   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6348   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6349   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6350   ( \l_tmpa_dim , \l_tmpb_dim ) --
6351   ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6352   \end { tikzpicture }
6353 }

```

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).

```

6354 \cs_new_protected:Npn \@@_draw_hlines:
6355 {
6356   \int_step_inline:nnn
6357   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6358   {
6359     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool

```

```

6360     \c@iRow
6361     { \int_eval:n { \c@iRow + 1 } }
6362   }
6363   {
6364     \tl_if_eq:NnF \l_@@_hlines_clist \c_@@_all_tl
6365     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6366     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6367   }
6368 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6369 \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.

```

6370 \cs_set:Npn \@@_Hline_i:n #1
6371 {
6372   \peek_remove_spaces:n
6373   {
6374     \peek_meaning:NTF \Hline
6375     { \@@_Hline_ii:nn { #1 + 1 } }
6376     { \@@_Hline_iii:n { #1 } }
6377   }
6378 }
6379 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6380 \cs_set:Npn \@@_Hline_iii:n #1
6381 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6382 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6383 {
6384   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6385   \skip_vertical:N \l_@@_rule_width_dim
6386   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6387   {
6388     \@@_hline:n
6389     {
6390       multiplicity = #1 ,
6391       position = \int_eval:n { \c@iRow + 1 } ,
6392       total-width = \dim_use:N \l_@@_rule_width_dim ,
6393       #2
6394     }
6395   }
6396   \egroup
6397 }

```

### 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`).

```

6398 \cs_new_protected:Npn \@@_custom_line:n #1
6399 {
6400   \str_clear_new:N \l_@@_command_str
6401   \str_clear_new:N \l_@@_ccommand_str
6402   \str_clear_new:N \l_@@_letter_str
6403   \tl_clear_new:N \l_@@_other_keys_tl
6404   \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.

```

6405 \bool_lazy_all:nTF
6406 {
6407   { \str_if_empty_p:N \l_@@_letter_str }
6408   { \str_if_empty_p:N \l_@@_command_str }
6409   { \str_if_empty_p:N \l_@@_ccommand_str }
6410 }
6411 { \@@_error:n { No~letter~and~no~command } }
6412 { \@@_custom_line_i:o \l_@@_other_keys_tl }
6413 }
6414 \keys_define:nm { nicematrix / custom-line }
6415 {
6416   letter .str_set:N = \l_@@_letter_str ,
6417   letter .value_required:n = true ,
6418   command .str_set:N = \l_@@_command_str ,
6419   command .value_required:n = true ,
6420   ccommand .str_set:N = \l_@@_cccommand_str ,
6421   ccommand .value_required:n = true ,
6422 }
6423 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6424 \cs_new_protected:Npn \@@_custom_line_i:n #1
6425 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6426 \bool_set_false:N \l_@@_tikz_rule_bool
6427 \bool_set_false:N \l_@@_dotted_rule_bool
6428 \bool_set_false:N \l_@@_color_bool
6429 \keys_set:nm { nicematrix / custom-line-bis } { #1 }
6430 \bool_if:NT \l_@@_tikz_rule_bool
6431 {
6432   \IfPackageLoadedF { tikz }
6433   { \@@_error:n { tikz-in-custom-line-without-tikz } }
6434   \bool_if:NT \l_@@_color_bool
6435   { \@@_error:n { color-in-custom-line-with-tikz } }
6436 }
6437 \bool_if:NT \l_@@_dotted_rule_bool
6438 {
6439   \int_compare:nNnT \l_@@_multiplicity_int > \c_one_int
6440   { \@@_error:n { key~multiplicity~with~dotted } }
6441 }
6442 \str_if_empty:NF \l_@@_letter_str
6443 {
6444   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6445   { \@@_error:n { Several~letters } }
6446   {
6447     \tl_if_in:NoTF
6448     \c_@@_forbidden_letters_str
6449     \l_@@_letter_str
6450     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6451   }

```

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.

```

6452 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str } ##1
6453 { \@@_v_custom_line:n { #1 } }
6454 }
6455 }
6456 }
6457 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6458 \str_if_empty:NF \l_@@_cccommand_str { \@@_c_custom_line:n { #1 } }
6459 }

```



```

6460 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|() []!@<> }
6461 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|() []!@<> }

```

The previous command `\@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{nicematrix/Rules}`). That's why the following set of keys has some keys which are no-op.

```

6462 \keys_define:nn { nicematrix / custom-line-bis }
6463 {
6464   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6465   multiplicity .initial:n = 1 ,
6466   multiplicity .value_required:n = true ,
6467   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6468   color .value_required:n = true ,
6469   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6470   tikz .value_required:n = true ,
6471   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6472   dotted .value_forbidden:n = true ,
6473   total-width .code:n = { } ,
6474   total-width .value_required:n = true ,
6475   width .code:n = { } ,
6476   width .value_required:n = true ,
6477   sep-color .code:n = { } ,
6478   sep-color .value_required:n = true ,
6479   unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
6480 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6481 \bool_new:N \l_@@_dotted_rule_bool
6482 \bool_new:N \l_@@_tikz_rule_bool
6483 \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`.

```

6484 \keys_define:nn { nicematrix / custom-line-width }
6485 {
6486   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6487   multiplicity .initial:n = 1 ,
6488   multiplicity .value_required:n = true ,
6489   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6490   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6491   \bool_set_true:N \l_@@_total_width_bool ,
6492   total-width .value_required:n = true ,
6493   width .meta:n = { total-width = #1 } ,
6494   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6495 }

```

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 `\@@_hline:n` (which is in the internal `\CodeAfter`).

```

6496 \cs_new_protected:Npn \@@_h_custom_line:n #1
6497 {

```

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`).

```

6498   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6499   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6500 }

```

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`).

```
6501 \cs_new_protected:Npn \@@_c_custom_line:n #1
6502 {
```

Here, we need an expandable command since it begins with an `\noalign`.

```
6503   \exp_args:Nc \NewExpandableDocumentCommand
6504     { nicematrix - \l_@@_ccommand_str }
6505     { 0 { } m }
6506     {
6507       \noalign
6508       {
6509         \@@_compute_rule_width:n { #1 , ##1 }
6510         \skip_vertical:n { \l_@@_rule_width_dim }
6511         \clist_map_inline:nn
6512           { ##2 }
6513           { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6514       }
6515     }
6516   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6517 }
```

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*.

```
6518 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6519 {
6520   \tl_if_in:nnTF { #2 } { - }
6521     { \@@_cut_on_hyphen:w #2 \q_stop }
6522     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6523   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6524     {
6525       \@@_hline:n
6526       {
6527         #1 ,
6528         start = \l_tmpa_tl ,
6529         end = \l_tmpb_tl ,
6530         position = \int_eval:n { \c@iRow + 1 } ,
6531         total-width = \dim_use:N \l_@@_rule_width_dim
6532       }
6533     }
6534 }

6535 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6536 {
6537   \bool_set_false:N \l_@@_tikz_rule_bool
6538   \bool_set_false:N \l_@@_total_width_bool
6539   \bool_set_false:N \l_@@_dotted_rule_bool
6540   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6541   \bool_if:NF \l_@@_total_width_bool
6542     {
6543       \bool_if:NTF \l_@@_dotted_rule_bool
6544         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6545         {
6546           \bool_if:NF \l_@@_tikz_rule_bool
6547             {
6548               \dim_set:Nn \l_@@_rule_width_dim
6549                 {
6550                   \arrayrulewidth * \l_@@_multiplicity_int
6551                   + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6552                 }
6553             }
6554         }
6555     }
6556 }
```

```

6557 \cs_new_protected:Npn \@@_v_custom_line:n #1
6558 {
6559   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6560   \tl_gput_right:Ne \g_@@_array_preamble_tl
6561     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6562   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6563     {
6564     \@@_vline:n
6565     {
6566       #1 ,
6567       position = \int_eval:n { \c@jCol + 1 } ,
6568       total-width = \dim_use:N \l_@@_rule_width_dim
6569     }
6570   }
6571   \@@_rec_preamble:n
6572 }

6573 \@@_custom_line:n
6574 { 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.

```

6575 \cs_new_protected:Npn \@@_test_hline_in_block:nnnn #1 #2 #3 #4 #5
6576 {
6577   \int_compare:nNnT \l_tmpa_tl > { #1 }
6578   {
6579     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6580     {
6581       \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6582       {
6583         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6584         { \bool_gset_false:N \g_tmpa_bool }
6585       }
6586     }
6587   }
6588 }

```

The same for vertical rules.

```

6589 \cs_new_protected:Npn \@@_test_vline_in_block:nnnn #1 #2 #3 #4 #5
6590 {
6591   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6592   {
6593     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6594     {
6595       \int_compare:nNnT \l_tmpb_tl > { #2 }
6596       {
6597         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6598         { \bool_gset_false:N \g_tmpa_bool }
6599       }
6600     }
6601   }
6602 }

6603 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6604 {
6605   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6606   {
6607     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6608     {

```

```

6609         \int_compare:nNnTF \l_tmpa_tl = { #1 }
6610         { \bool_gset_false:N \g_tmpa_bool }
6611         {
6612             \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6613             { \bool_gset_false:N \g_tmpa_bool }
6614         }
6615     }
6616 }
6617 }

6618 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6619 {
6620     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6621     {
6622         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6623         {
6624             \int_compare:nNnTF \l_tmpb_tl = { #2 }
6625             { \bool_gset_false:N \g_tmpa_bool }
6626             {
6627                 \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6628                 { \bool_gset_false:N \g_tmpa_bool }
6629             }
6630         }
6631     }
6632 }

```

## 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.

```

6633 \cs_new_protected:Npn \@@_compute_corners:
6634 {
6635     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6636     { \@@_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`).

```

6637     \clist_clear:N \l_@@_corners_cells_clist
6638     \clist_map_inline:Nn \l_@@_corners_cells_clist
6639     {
6640         \str_case:nnF { ##1 }
6641         {
6642             { NW }
6643             { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6644             { NE }
6645             { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6646             { SW }
6647             { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6648             { SE }
6649             { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6650         }
6651         { \@@_error:nn { bad-corner } { ##1 } }
6652     }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6653     \clist_if_empty:NF \l_@@_corners_cells_clist
6654     {

```

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.

```

6655     \tl_gput_right:Nc \g_@@_aux_tl
6656     {
6657         \cs_set_nopar:Npn \exp_not:N \l_@@_corners_cells_clist
6658         { \l_@@_corners_cells_clist }
6659     }
6660 }
6661 }

```

```

6662 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6663 {
6664     \int_step_inline:nnn { #1 } { #3 }
6665     {
6666         \int_step_inline:nnn { #2 } { #4 }
6667         { \cs_set_nopar:cpn { @@ _ block _ ##1 - ###1 } { } }
6668     }
6669 }

```

```

6670 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6671 {
6672     \cs_if_exist:cTF
6673     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6674     \prg_return_true:
6675     \prg_return_false:
6676 }

```

“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.

```

6677 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6678 {

```

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.

```

6679     \bool_set_false:N \l_tmpa_bool
6680     \int_zero_new:N \l_@@_last_empty_row_int
6681     \int_set:Nn \l_@@_last_empty_row_int { #1 }
6682     \int_step_inline:nnnn { #1 } { #3 } { #5 }
6683     {
6684         \bool_lazy_or:nnTF
6685         {
6686             \cs_if_exist_p:c
6687             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6688         }
6689         { \@@_if_in_block_p:nn { ##1 } { #2 } }
6690         { \bool_set_true:N \l_tmpa_bool }
6691     }

```

```

6692         \bool_if:NF \l_tmpa_bool
6693         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6694     }
6695 }

```

Now, you determine the last empty cell in the row of number 1.

```

6696     \bool_set_false:N \l_tmpa_bool
6697     \int_zero_new:N \l_@@_last_empty_column_int
6698     \int_set:Nn \l_@@_last_empty_column_int { #2 }
6699     \int_step_inline:nnnn { #2 } { #4 } { #6 }
6700     {
6701         \bool_lazy_or:nnTF
6702         {
6703             \cs_if_exist_p:c
6704             { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6705         }
6706         { \@@_if_in_block_p:nn { #1 } { ##1 } }
6707         { \bool_set_true:N \l_tmpa_bool }
6708     }
6709     \bool_if:NF \l_tmpa_bool
6710     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6711 }
6712 }

```

Now, we loop over the rows.

```

6713     \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6714     {

```

We treat the row number ##1 with another loop.

```

6715         \bool_set_false:N \l_tmpa_bool
6716         \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6717         {
6718             \bool_lazy_or:nnTF
6719             { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6720             { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6721             { \bool_set_true:N \l_tmpa_bool }
6722         }
6723         \bool_if:NF \l_tmpa_bool
6724         {
6725             \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6726             \clist_put_right:Nn
6727             \l_@@_corners_cells_clist
6728             { ##1 - #####1 }
6729             \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6730         }
6731     }
6732 }
6733 }
6734 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6735 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6736 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }

```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient: `\clist_if_in:NeT \l_@@_corners_cells_clist { #1 } ...`

## 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.

```
6737 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```
6738 \keys_define:nn { nicematrix / NiceMatrixBlock }
6739 {
6740   auto-columns-width .code:n =
6741   {
6742     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6743     \dim_gzero_new:N \g_@@_max_cell_width_dim
6744     \bool_set_true:N \l_@@_auto_columns_width_bool
6745   }
6746 }

6747 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6748 {
6749   \int_gincr:N \g_@@_NiceMatrixBlock_int
6750   \dim_zero:N \l_@@_columns_width_dim
6751   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6752   \bool_if:NT \l_@@_block_auto_columns_width_bool
6753   {
6754     \cs_if_exist:cT
6755       { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6756       {
6757         \dim_set:Nn \l_@@_columns_width_dim
6758         {
6759           \use:c
6760             { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6761         }
6762       }
6763   }
6764 }
```

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`).

```
6765 {
6766   \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.

```
6767   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6768   {
6769     \bool_if:NT \l_@@_block_auto_columns_width_bool
6770     {
6771       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6772       \iow_shipout:Ne \@mainaux
6773       {
6774         \cs_gset:cpn
6775           { @@_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.

```
6776           { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6777       }
6778       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6779     }
6780   }
6781   \ignorespacesafterend
6782 }
```

## 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).

```

6783 \cs_new_protected:Npn \@@_create_extra_nodes:
6784 {
6785   \bool_if:nTF \l_@@_medium_nodes_bool
6786   {
6787     \bool_if:NTF \l_@@_large_nodes_bool
6788     \@@_create_medium_and_large_nodes:
6789     \@@_create_medium_nodes:
6790   }
6791   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6792 }

```

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`.

```

6793 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6794 {
6795   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6796   {
6797     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6798     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6799     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6800     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6801   }
6802   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6803   {
6804     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6805     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6806     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6807     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6808   }

```

We begin the two nested loops over the rows and the columns of the array.

```

6809   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6810   {
6811     \int_step_variable:nnNn
6812     \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.

```

6813     {
6814         \cs_if_exist:cT
6815         { 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`.

```

6816     {
6817         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6818         \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
6819         { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6820         \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6821         {
6822             \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6823             { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6824         }

```

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`.

```

6825         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6826         \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
6827         { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
6828         \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6829         {
6830             \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
6831             { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6832         }
6833     }
6834 }
6835 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6836 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6837 {
6838     \dim_compare:nNnT
6839     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
6840     {
6841         \@@_qpoint:n { row - \@@_i: - base }
6842         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
6843         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
6844     }
6845 }
6846 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6847 {
6848     \dim_compare:nNnT
6849     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
6850     {
6851         \@@_qpoint:n { col - \@@_j: }
6852         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
6853         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
6854     }
6855 }
6856 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6857 \cs_new_protected:Npn \@@_create_medium_nodes:
6858 {
6859     \pgfpicture
6860     \pgfrememberpicturepositiononpagetrue
6861     \pgf@relevantforpicturesizefalse
6862     \@@_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”.

```

6863     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6864     \@@_create_nodes:
6865     \endpgfpicture
6866 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones<sup>14</sup>. 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:`.

```

6867 \cs_new_protected:Npn \@@_create_large_nodes:
6868 {
6869   \pgfpicture
6870   \pgfrememberpicturepositiononpagetrue
6871   \pgf@relevantforpicturesizefalse
6872   \@@_computations_for_medium_nodes:
6873   \@@_computations_for_large_nodes:
6874   \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6875   \@@_create_nodes:
6876   \endpgfpicture
6877 }

6878 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6879 {
6880   \pgfpicture
6881   \pgfrememberpicturepositiononpagetrue
6882   \pgf@relevantforpicturesizefalse
6883   \@@_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”.

```

6884     \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
6885     \@@_create_nodes:
6886     \@@_computations_for_large_nodes:
6887     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6888     \@@_create_nodes:
6889     \endpgfpicture
6890 }

```

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.

```

6891 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6892 {
6893   \int_set_eq:NN \l_@@_first_row_int \c_one_int
6894   \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`.

```

6895   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6896   {
6897     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6898     {
6899       (
6900         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6901         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6902       )
6903       / 2
6904     }

```

---

<sup>14</sup>If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6905     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6906     { l_@@_row_ \@@_i: _ min_dim }
6907   }
6908   \int_step_variable:nnN { \c@jCol - 1 } \@@_j:
6909   {
6910     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6911     {
6912       (
6913         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6914         \dim_use:c
6915         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6916       )
6917       / 2
6918     }
6919     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6920     { l_@@_column _ \@@_j: _ max _ dim }
6921   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6922   \dim_sub:cn
6923   { l_@@_column _ 1 _ min _ dim }
6924   \l_@@_left_margin_dim
6925   \dim_add:cn
6926   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6927   \l_@@_right_margin_dim
6928 }

```

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`).

```

6929 \cs_new_protected:Npn \@@_create_nodes:
6930 {
6931   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6932   {
6933     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6934     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

6935     \@@_pgf_rect_node:nnnnn
6936     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6937     { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
6938     { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
6939     { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
6940     { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
6941     \str_if_empty:NF \l_@@_name_str
6942     {
6943       \pgfnodealias
6944       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6945       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6946     }
6947   }
6948 }

```

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  $n$ .

```

6949   \seq_map_pairwise_function:NNN
6950   \g_@@_multicolumn_cells_seq
6951   \g_@@_multicolumn_sizes_seq
6952   \@@_node_for_multicolumn:nn
6953 }

```

```

6954 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6955 {
6956   \cs_set_nopar:Npn \@@_i: { #1 }
6957   \cs_set_nopar:Npn \@@_j: { #2 }
6958 }

```

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”).

```

6959 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6960 {
6961   \@@_extract_coords_values: #1 \q_stop
6962   \@@_pgf_rect_node:nnnnn
6963   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6964   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
6965   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
6966   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
6967   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
6968   \str_if_empty:NF \l_@@_name_str
6969   {
6970     \pgfnodealias
6971     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6972     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
6973   }
6974 }

```

## 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).

```

6975 \keys_define:mn { nicematrix / Block / FirstPass }
6976 {
6977   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
6978     \bool_set_true:N \l_@@_p_block_bool ,
6979   j .value_forbidden:n = true ,
6980   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6981   l .value_forbidden:n = true ,
6982   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6983   r .value_forbidden:n = true ,
6984   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6985   c .value_forbidden:n = true ,
6986   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6987   L .value_forbidden:n = true ,
6988   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6989   R .value_forbidden:n = true ,
6990   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6991   C .value_forbidden:n = true ,
6992   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
6993   t .value_forbidden:n = true ,
6994   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
6995   T .value_forbidden:n = true ,
6996   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
6997   b .value_forbidden:n = true ,
6998   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
6999   B .value_forbidden:n = true ,

```

```

7000 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7001 m .value_forbidden:n = true ,
7002 v-center .meta:n = m ,
7003 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7004 p .value_forbidden:n = true ,
7005 color .code:n =
7006   \@@_color:n { #1 }
7007   \tl_set_rescan:Nnn
7008     \l_@@_draw_tl
7009     { \char_set_catcode_other:N ! }
7010     { #1 } ,
7011 color .value_required:n = true ,
7012 respect-arraystretch .code:n =
7013   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7014 respect-arraystretch .value_forbidden:n = true ,
7015 }

```

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.

```

7016 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7017 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7018 {

```

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).

```

7019   \peek_remove_spaces:n
7020   {
7021     \tl_if_blank:nTF { #2 }
7022       { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7023       {
7024         \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7025         \@@_Block_i_czech \@@_Block_i
7026         #2 \q_stop
7027       }
7028     { #1 } { #3 } { #4 }
7029   }
7030 }

```

With the following construction, we extract the values of  $i$  and  $j$  in the first mandatory argument of the command.

```

7031 \cs_new:Npn \@@_Block_i #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`.

```

7032 {
7033   \char_set_catcode_active:N -
7034   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7035 }

```

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.

```

7036 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7037 {

```

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).

```

7038   \bool_lazy_or:nnTF
7039     { \tl_if_blank_p:n { #1 } }
7040     { \str_if_eq_p:ee { * } { #1 } }
7041     { \int_set:Nn \l_tmpa_int { 100 } }
7042     { \int_set:Nn \l_tmpa_int { #1 } }
7043   \bool_lazy_or:nnTF
7044     { \tl_if_blank_p:n { #2 } }
7045     { \str_if_eq_p:ee { * } { #2 } }
7046     { \int_set:Nn \l_tmpb_int { 100 } }
7047     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7048   \int_compare:nNnTF \l_tmpb_int = \c_one_int
7049     {
7050       \tl_if_empty:NTF \l_@@_hpos_cell_tl
7051         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7052         { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7053     }
7054     { \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.

```

7055   \keys_set_known:n { nicematrix / Block / FirstPass } { #3 }
7056   \tl_set:Ne \l_tmpa_tl
7057     {
7058       { \int_use:N \c@iRow }
7059       { \int_use:N \c@jCol }
7060       { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7061       { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7062     }

```

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).

```

7063   \bool_set_false:N \l_tmpa_bool
7064   \bool_if:NT \l_@@_amp_in_blocks_bool

```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

7065     { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7066   \bool_case:nF
7067     {
7068       \l_tmpa_bool                               { \@@_Block_vii:eennn }
7069       \l_@@_p_block_bool                         { \@@_Block_vi:eennn }

```

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.

```

7070     \l_@@_X_bool                               { \@@_Block_v:eennn }
7071     { \tl_if_empty_p:n { #5 } }                 { \@@_Block_v:eennn }
7072     { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7073     { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7074     }
7075     { \@@_Block_v:eennn }
7076   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7077 }

```

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.

```

7078 \cs_generate_variant:Nn \@@_Block_iv:nnnnn { e e }
7079 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7080 {
7081   \int_gincr:N \g_@@_block_box_int
7082   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7083   {
7084     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7085     {
7086       \@@_actually_diagbox:nnnnnn
7087       { \int_use:N \c@iRow }
7088       { \int_use:N \c@jCol }
7089       { \int_eval:n { \c@iRow + #1 - 1 } }
7090       { \int_eval:n { \c@jCol + #2 - 1 } }
7091       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7092       { \g_@@_row_style_tl \exp_not:n { ##2 } }
7093     }
7094   }
7095   \box_gclear_new:c
7096   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7097   \hbox_gset:cn
7098   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
7099   {

```

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` with `\RequirePackage{expl3}`).

```

7100   \tl_if_empty:NTF \l_@@_color_tl
7101   { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7102   { \@@_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`.

```

7103   \int_compare:nNnT { #1 } = \c_one_int
7104   {
7105     \int_if_zero:nTF \c@iRow
7106     {

```

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}$

```

```

7107         \cs_set_eq:NN \Block \@@_NullBlock:
7108         \l_@@_code_for_first_row_tl
7109     }
7110     {
7111         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7112         {
7113             \cs_set_eq:NN \Block \@@_NullBlock:
7114             \l_@@_code_for_last_row_tl
7115         }
7116     }
7117     \g_@@_row_style_tl
7118 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7119     \@@_reset_arraystretch:
7120     \dim_zero:N \extrarowheight

```

**#4** is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7121     #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.).

```

7122     \@@_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}`.

```

7123     \bool_if:NTF \l_@@_tabular_bool
7124     {
7125         \bool_lazy_all:nTF
7126         {
7127             { \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.

```

7128         { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7129         { ! \g_@@_rotate_bool }
7130     }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7131     {
7132         \use:e
7133     {

```

The `\exp_not:N` is mandatory before `\begin`.

```

7134         \exp_not:N \begin { minipage }%
7135         [ \str_lowercase:o \l_@@_vpos_block_str ]
7136         { \l_@@_col_width_dim }
7137         \str_case:on \l_@@_hpos_block_str
7138         { c \centering r \raggedleft l \raggedright }
7139     }

```



```

7140         #5
7141         \end { minipage }
7142     }

```

In the other cases, we use a `{tabular}`.

```

7143     {
7144     \use:e
7145     {
7146     \exp_not:N \begin { tabular }%
7147     [ \str_lowercase:o \l_@@_vpos_block_str ]
7148     { @ { } \l_@@_hpos_block_str @ { } }
7149     }
7150     #5
7151     \end { tabular }
7152     }
7153 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7154     {
7155     \c_math_toggle_token
7156     \use:e
7157     {
7158     \exp_not:N \begin { array }%
7159     [ \str_lowercase:o \l_@@_vpos_block_str ]
7160     { @ { } \l_@@_hpos_block_str @ { } }
7161     }
7162     #5
7163     \end { array }
7164     \c_math_toggle_token
7165     }
7166 }

```

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).

```

7167     \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.

```

7168     \int_compare:nNnT { #2 } = \c_one_int
7169     {
7170     \dim_gset:Nn \g_@@_blocks_wd_dim
7171     {
7172     \dim_max:nn
7173     \g_@@_blocks_wd_dim
7174     {
7175     \box_wd:c
7176     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7177     }
7178     }
7179 }

```

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.

```

7180     \bool_lazy_and:nnT
7181     { \int_compare_p:nNn { #1 } = \c_one_int }

```

If the user has not used a key for the vertical position of the block, then `\l_@@_vpos_block_str` remains empty.

```

7182     { \str_if_empty_p:N \l_@@_vpos_block_str }
7183     {
7184     \dim_gset:Nn \g_@@_blocks_ht_dim

```

```

7185     {
7186         \dim_max:nn
7187         \g_@@_blocks_ht_dim
7188         {
7189             \box_ht:c
7190             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7191         }
7192     }
7193 \dim_gset:Nn \g_@@_blocks_dp_dim
7194 {
7195     \dim_max:nn
7196     \g_@@_blocks_dp_dim
7197     {
7198         \box_dp:c
7199         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7200     }
7201 }
7202 }
7203 \seq_gput_right:Ne \g_@@_blocks_seq
7204 {
7205     \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.

```

7206     {
7207         \exp_not:n { #3 } ,
7208         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7209     \bool_if:NT \g_@@_rotate_bool
7210     {
7211         \bool_if:NTF \g_@@_rotate_c_bool
7212         { m }
7213         { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7214     }
7215 }
7216 {
7217     \box_use_drop:c
7218     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7219 }
7220 }
7221 \bool_set_false:N \g_@@_rotate_c_bool
7222 }

7223 \cs_new:Npn \@@_adjust_hpos_rotate:
7224 {
7225     \bool_if:NT \g_@@_rotate_bool
7226     {
7227         \str_set:Ne \l_@@_hpos_block_str
7228         {
7229             \bool_if:NTF \g_@@_rotate_c_bool
7230             { c }
7231             {
7232                 \str_case:onF \l_@@_vpos_block_str
7233                 { b l B l t r T r }
7234                 { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7235             }
7236         }
7237     }
7238 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7239 \cs_new_protected:Npn \@@_rotate_box_of_block:
7240 {
7241   \box_grotate:cn
7242   { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7243   { 90 }
7244   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7245   {
7246     \vbox_gset_top:cn
7247     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7248     {
7249       \skip_vertical:n { 0.8 ex }
7250       \box_use:c
7251       { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7252     }
7253   }
7254   \bool_if:NT \g_@@_rotate_c_bool
7255   {
7256     \hbox_gset:cn
7257     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7258     {
7259       \c_math_toggle_token
7260       \vcenter
7261       {
7262         \box_use:c
7263         { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7264       }
7265       \c_math_toggle_token
7266     }
7267   }
7268 }

```

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:nnnnn).

#1 is  $i$  (the number of rows of the block), #2 is  $j$  (the number of columns of the block), #3 is the list of key=values pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7269 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }
7270 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7271 {
7272   \seq_gput_right:Ne \g_@@_blocks_seq
7273   {
7274     \l_tmpa_tl
7275     { \exp_not:n { #3 } }
7276     {
7277       \bool_if:NTF \l_@@_tabular_bool
7278       {
7279         \group_begin:

```

The following command will be no-op when respect-arraystretch is in force.

```

7280   \@@_reset_arraystretch:
7281   \exp_not:n
7282   {
7283     \dim_zero:N \extrarowheight
7284     #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).

```

7285         \bool_if:NT \c_@@_testphase_table_bool
7286         { \tag_stop:n { table } }
7287         \use:e
7288         {
7289             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7290             { @ { } \l_@@_hpos_block_str @ { } }
7291         }
7292         #5
7293         \end { tabular }
7294     }
7295     \group_end:
7296 }

```

When we are *not* in an environment {NiceTabular} (or similar).

```

7297     {
7298         \group_begin:

```

The following will be no-op when respect-arraystretch is in force.

```

7299         \@@_reset_arraystretch:
7300         \exp_not:n
7301         {
7302             \dim_zero:N \extrarowheight
7303             #4
7304             \c_math_toggle_token
7305             \use:e
7306             {
7307                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7308                 { @ { } \l_@@_hpos_block_str @ { } }
7309             }
7310             #5
7311             \end { array }
7312             \c_math_toggle_token
7313         }
7314         \group_end:
7315     }
7316 }
7317 }
7318 }

```

The following macro is for the case of a \Block which uses the key p.

```

7319 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }
7320 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7321 {
7322     \seq_gput_right:Ne \g_@@_blocks_seq
7323     {
7324         \l_tmpa_tl
7325         { \exp_not:n { #3 } }
7326         {
7327             \group_begin:
7328             \exp_not:n { #4 #5 }
7329             \group_end:
7330         }
7331     }
7332 }

```

The following macro is for the case of a \Block which uses the key p.

```

7333 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }
7334 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7335 {
7336     \seq_gput_right:Ne \g_@@_blocks_seq
7337     {

```

```

7338     \l_tmpa_tl
7339     { \exp_not:n { #3 } }
7340     { \exp_not:n { #4 #5 } }
7341   }
7342 }

```

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).

```

7343 \keys_define:nn { nicematrix / Block / SecondPass }
7344 {
7345   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7346   ampersand-in-blocks .default:n = true ,
7347   &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7348   tikz .code:n =
7349     \IfPackageLoadedTF { tikz }
7350     { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7351     { \@@_error:n { tikz~key~without~tikz } } ,
7352   tikz .value_required:n = true ,
7353   fill .code:n =
7354     \tl_set_rescan:Nnn
7355     \l_@@_fill_tl
7356     { \char_set_catcode_other:N ! }
7357     { #1 } ,
7358   fill .value_required:n = true ,
7359   opacity .tl_set:N = \l_@@_opacity_tl ,
7360   opacity .value_required:n = true ,
7361   draw .code:n =
7362     \tl_set_rescan:Nnn
7363     \l_@@_draw_tl
7364     { \char_set_catcode_other:N ! }
7365     { #1 } ,
7366   draw .default:n = default ,
7367   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7368   rounded-corners .default:n = 4 pt ,
7369   color .code:n =
7370     \@@_color:n { #1 }
7371     \tl_set_rescan:Nnn
7372     \l_@@_draw_tl
7373     { \char_set_catcode_other:N ! }
7374     { #1 } ,
7375   borders .clist_set:N = \l_@@_borders_clist ,
7376   borders .value_required:n = true ,
7377   hvlines .meta:n = { vlines , hlines } ,
7378   vlines .bool_set:N = \l_@@_vlines_block_bool ,
7379   vlines .default:n = true ,
7380   hlines .bool_set:N = \l_@@_hlines_block_bool ,
7381   hlines .default:n = true ,
7382   line-width .dim_set:N = \l_@@_line_width_dim ,
7383   line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7384   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7385     \bool_set_true:N \l_@@_p_block_bool ,
7386   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7387   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7388   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7389   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7390     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7391   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7392     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,

```

```

7393 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7394         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7395 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7396 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7397 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7398 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7399 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7400 m .value_forbidden:n = true ,
7401 v-center .meta:n = m ,
7402 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7403 p .value_forbidden:n = true ,
7404 name .tl_set:N = \l_@@_block_name_str ,
7405 name .value_required:n = true ,
7406 name .initial:n = ,
7407 respect-arraystretch .code:n =
7408     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7409 respect-arraystretch .value_forbidden:n = true ,
7410 transparent .bool_set:N = \l_@@_transparent_bool ,
7411 transparent .default:n = true ,
7412 transparent .initial:n = false ,
7413 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7414 }

```

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.

```

7415 \cs_new_protected:Npn \@@_draw_blocks:
7416 {
7417     \bool_if:NTF \c_@@_tagging_array_bool
7418     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7419     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7420     \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7421 }
7422 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }
7423 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7424 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7425     \int_zero_new:N \l_@@_last_row_int
7426     \int_zero_new: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: 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.

```

7427     \int_compare:nNnTF { #3 } > { 99 }
7428     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7429     { \int_set:Nn \l_@@_last_row_int { #3 } }
7430     \int_compare:nNnTF { #4 } > { 99 }
7431     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7432     { \int_set:Nn \l_@@_last_col_int { #4 } }
7433     \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7434     {
7435         \bool_lazy_and:nnTF
7436         \l_@@_preamble_bool
7437         {
7438             \int_compare_p:n
7439             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7440         }

```

```

7441     {
7442     \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7443     \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7444     \@@_msg_redirect_name:nn { columns-not-used } { none }
7445     }
7446     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7447   }
7448   {
7449   \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7450   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7451   {
7452     \@@_Block_v:nneenn
7453     { #1 }
7454     { #2 }
7455     { \int_use:N \l_@@_last_row_int }
7456     { \int_use:N \l_@@_last_col_int }
7457     { #5 }
7458     { #6 }
7459   }
7460 }
7461 }

```

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

```

7462 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7463 {

```

The group is for the keys.

```

7464   \group_begin:
7465   \int_compare:nNnT { #1 } = { #3 }
7466   { \str_set:Nn \l_@@_vpos_block_str { t } }
7467   \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`.

```

7468   \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7469   \bool_lazy_and:nnT
7470   \l_@@_vlines_block_bool
7471   { ! \l_@@_ampersand_bool }
7472   {
7473     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7474     {
7475       \@@_vlines_block:nnn
7476       { \exp_not:n { #5 } }
7477       { #1 - #2 }
7478       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7479     }
7480   }
7481   \bool_if:NT \l_@@_hlines_block_bool
7482   {
7483     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7484     {
7485       \@@_hlines_block:nnn
7486       { \exp_not:n { #5 } }
7487       { #1 - #2 }
7488       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7489     }
7490   }
7491   \bool_if:NF \l_@@_transparent_bool
7492   {
7493     \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7494     {

```

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).

```

7495     \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7496     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7497   }
7498 }

```

```

7499 \tl_if_empty:NF \l_@@_draw_tl
7500 {
7501   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7502   { \@@_error:n { hlines~with~color } }
7503   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7504   {
7505     \@@_stroke_block:nnn

```

#5 are the options

```

7506     { \exp_not:n { #5 } }
7507     { #1 - #2 }
7508     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7509   }
7510   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7511   { { #1 } { #2 } { #3 } { #4 } }
7512 }

7513 \clist_if_empty:NF \l_@@_borders_clist
7514 {
7515   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7516   {
7517     \@@_stroke_borders_block:nnn
7518     { \exp_not:n { #5 } }
7519     { #1 - #2 }
7520     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7521   }
7522 }

7523 \tl_if_empty:NF \l_@@_fill_tl
7524 {
7525   \tl_if_empty:NF \l_@@_opacity_tl
7526   {
7527     \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
7528       {
7529         \tl_set:Ne \l_@@_fill_tl
7530         {
7531           [ opacity = \l_@@_opacity_tl ,
7532             \tl_tail:o \l_@@_fill_tl
7533         ]
7534       }
7535     ]
7536     \tl_set:Ne \l_@@_fill_tl
7537     { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
7538   }
7539 }

7540 \tl_gput_right:Ne \g_@@_pre_code_before_tl
7541 {
7542   \exp_not:N \roundedrectanglecolor
7543   \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
7544     { \l_@@_fill_tl }
7545     { { \l_@@_fill_tl } }
7546     { #1 - #2 }
7547     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7548     { \dim_use:N \l_@@_rounded_corners_dim }
7549   }
7550 }

```



```

7551 \seq_if_empty:NF \l_@@_tikz_seq
7552 {
7553   \tl_gput_right:Ne \g_nicematrix_code_before_tl
7554   {
7555     \@@_block_tikz:nnnnn
7556     { \seq_use:Nn \l_@@_tikz_seq { , } }
7557     { #1 }
7558     { #2 }
7559     { \int_use:N \l_@@_last_row_int }
7560     { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of list of Tikz keys.

```

7561   }
7562 }

7563 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7564 {
7565   \tl_gput_right:Ne \g_@@_pre_code_after_tl
7566   {
7567     \@@_actually_diagbox:nnnnnn
7568     { #1 }
7569     { #2 }
7570     { \int_use:N \l_@@_last_row_int }
7571     { \int_use:N \l_@@_last_col_int }
7572     { \exp_not:n { ##1 } }
7573     { \exp_not:n { ##2 } }
7574   }
7575 }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} & & one & \\\
& & & two & \\\
three & & four & five & \\\
six & & seven & eight & \\\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block		one
three	four	two
six	seven	five
		eight

We highlight the node `1-1-block-short`

our block	one
three	two
six	five
	eight

The construction of the node corresponding to the merged cells.

```

7576 \pgfpicture
7577 \pgfrememberpicturepositiononpagetrue
7578 \pgf@relevantforpicturesizefalse
7579 \@@_qpoint:n { row - #1 }
7580 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7581 \@@_qpoint:n { col - #2 }
7582 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7583 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7584 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7585 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7586 \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.

```

7587 \@@_pgf_rect_node:nnnnn
7588   { \@@_env: - #1 - #2 - block }
7589   \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7590 \str_if_empty:NF \l_@@_block_name_str
7591   {
7592     \pgfnodealias
7593     { \@@_env: - \l_@@_block_name_str }
7594     { \@@_env: - #1 - #2 - block }
7595     \str_if_empty:NF \l_@@_name_str
7596     {
7597       \pgfnodealias
7598       { \l_@@_name_str - \l_@@_block_name_str }
7599       { \@@_env: - #1 - #2 - block }
7600     }
7601   }

```

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.

```

7602 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7603   {
7604     \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.

```

7605     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7606     {

```

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.

```

7607       \cs_if_exist:cT
7608       { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7609       {
7610         \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7611         {
7612           \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7613           \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7614         }
7615       }
7616     }

```

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.

```

7617     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7618     {
7619       \@@_qpoint:n { col - #2 }
7620       \dim_set_eq:NN \l_tmpb_dim \pgf@x
7621     }
7622     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7623     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7624     {
7625       \cs_if_exist:cT
7626       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7627       {
7628         \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7629         {
7630           \pgfpointanchor
7631           { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7632           { east }

```

```

7633         \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7634     }
7635 }
7636 }
7637 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7638 {
7639     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7640     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7641 }
7642 \@@_pgf_rect_node:nnnnn
7643 { \@@_env: - #1 - #2 - block - short }
7644 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7645 }

```

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.

```

7646 \bool_if:NT \l_@@_medium_nodes_bool
7647 {
7648     \@@_pgf_rect_node:nnn
7649     { \@@_env: - #1 - #2 - block - medium }
7650     { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
7651     {
7652         \pgfpointanchor
7653         { \@@_env:
7654             - \int_use:N \l_@@_last_row_int
7655             - \int_use:N \l_@@_last_col_int - medium
7656         }
7657         { south~east }
7658     }
7659 }
7660 \endpgfpicture

7661 \bool_if:NTF \l_@@_ampersand_bool
7662 {
7663     \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7664     \int_zero_new:N \l_@@_split_int
7665     \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7666     \pgfpicture
7667     \pgfrememberpicturerepositiononpagetrue
7668     \pgf@relevantforpicturesizefalse
7669
7670     \@@_qpoint:n { row - #1 }
7671     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7672     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7673     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7674     \@@_qpoint:n { col - #2 }
7675     \dim_set_eq:NN \l_tmpa_dim \pgf@x
7676     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7677     \dim_set:Nn \l_tmpb_dim
7678     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7679     \bool_lazy_or:nnT
7680     \l_@@_vlines_block_bool
7681     { \tl_if_eq_p:NN \l_@@_vlines_clist \c_@@_all_tl }
7682     {
7683         \int_step_inline:nn { \l_@@_split_int - 1 }
7684         {
7685             \pgfpathmoveto
7686             {
7687                 \pgfpoint
7688                 { \l_tmpa_dim + ##1 \l_tmpb_dim }
7689                 \l_@@_tmpc_dim
7690             }
7691             \pgfpathlineto

```

```

7692         {
7693             \pgfpoint
7694             { \l_tmpa_dim + ##1 \l_tmpb_dim }
7695             \l_@@_tmpd_dim
7696         }
7697         \CT@arc@
7698         \pgfsetlinewidth { 1.1 \arrayrulewidth }
7699         \pgfsetrectcap
7700         \pgfusepathqstroke
7701     }
7702 }
7703 \@@_qpoint:n { row - #1 - base }
7704 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7705 \int_step_inline:nn \l_@@_split_int
7706 {
7707     \group_begin:
7708     \dim_set:Nn \col@sep
7709     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
7710     \pgftransformshift
7711     {
7712         \pgfpoint
7713         {
7714             \l_tmpa_dim + ##1 \l_tmpb_dim -
7715             \str_case:on \l_@@_hpos_block_str
7716             {
7717                 l { \l_tmpb_dim + \col@sep }
7718                 c { 0.5 \l_tmpb_dim }
7719                 r { \col@sep }
7720             }
7721         }
7722         { \l_@@_tmpc_dim }
7723     }
7724     \pgfset { inner~sep = \c_zero_dim }
7725     \pgfnode
7726     { rectangle }
7727     {
7728         \str_case:on \l_@@_hpos_block_str
7729         {
7730             c { base }
7731             l { base-west }
7732             r { base-east }
7733         }
7734     }
7735     { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
7736     \group_end:
7737 }
7738 \endpgfpicture
7739 }

```

Now the case where there is no ampersand & in the content of the block.

```

7740 {
7741     \bool_if:NTF \l_@@_p_block_bool
7742     {

```

When the final user has used the key p, we have to compute the width.

```

7743     \pgfpicture
7744     \pgfrememberpicturepositiononpagetrue
7745     \pgf@relevantforpicturesizefalse
7746     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7747     {
7748         \@@_qpoint:n { col - #2 }
7749         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7750         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7751     }

```

```

7752         {
7753             \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7754             \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7755             \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7756         }
7757         \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7758     \endpgfpicture
7759     \hbox_set:Nn \l_@@_cell_box
7760     {
7761         \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7762             { \g_tmpb_dim }
7763             \str_case:on \l_@@_hpos_block_str
7764                 { c \centering r \raggedleft l \raggedright j { } }
7765             #6
7766         \end { minipage }
7767     }
7768 }
7769 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7770 \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.

```

7771     \pgfpicture
7772     \pgfrememberpicturepositiononpagetrue
7773     \pgf@relevantforpicturesizefalse
7774     \bool_lazy_any:nTF
7775     {
7776         { \str_if_empty_p:N \l_@@_vpos_block_str } % added 2024/06/29
7777         { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7778         { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7779         { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7780     }
7781     {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7782         \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`.

```

7783         \bool_if:nT \g_@@_last_col_found_bool
7784         {
7785             \int_compare:nNnT { #2 } = \g_@@_col_total_int
7786             { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7787         }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7788         \tl_set:Ne \l_tmpa_tl
7789         {
7790             \str_case:on \l_@@_vpos_block_str
7791             {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7792             { } { % added 2024-06-29
7793                 \str_case:on \l_@@_hpos_block_str
7794                 {
7795                     c { center }
7796                     l { west }
7797                     r { east }
7798                     j { center }
7799                 }
7800             }
7801         c {
7802             \str_case:on \l_@@_hpos_block_str

```

```

7803         {
7804             c { center }
7805             l { west }
7806             r { east }
7807             j { center }
7808         }
7809     }
7810 }
7811 T {
7812     \str_case:on \l_@@_hpos_block_str
7813     {
7814         c { north }
7815         l { north-west }
7816         r { north-east }
7817         j { north }
7818     }
7819 }
7820 }
7821 B {
7822     \str_case:on \l_@@_hpos_block_str
7823     {
7824         c { south }
7825         l { south-west }
7826         r { south-east }
7827         j { south }
7828     }
7829 }
7830 }
7831 }
7832 }
7833 \pgftransformshift
7834 {
7835     \pgfpointanchor
7836     {
7837         \@@_env: - #1 - #2 - block
7838         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7839     }
7840     { \l_tmpa_tl }
7841 }
7842 \pgfset { inner-sep = \c_zero_dim }
7843 \pgfnode
7844 { rectangle }
7845 { \l_tmpa_tl }
7846 { \box_use_drop:N \l_@@_cell_box } { } { }
7847 }

```

End of the case when  $\l_@@_vpos\_block\_str$  is equal to c, T or B. Now, the other cases.

```

7848 {
7849     \pgfextracty \l_tmpa_dim
7850     {
7851         \@@_qpoint:n
7852         {
7853             row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7854             - base
7855         }
7856     }
7857     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in  $\pgf@x$ ) the  $x$ -value of the center of the block.

```

7858 \pgfpointanchor
7859 {
7860     \@@_env: - #1 - #2 - block
7861     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }

```

```

7862     }
7863     {
7864         \str_case:on \l_@@_hpos_block_str
7865         {
7866             c { center }
7867             l { west }
7868             r { east }
7869             j { center }
7870         }
7871     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7872         \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7873         \pgfset { inner~sep = \c_zero_dim }
7874         \pgfnode
7875         { rectangle }
7876         {
7877             \str_case:on \l_@@_hpos_block_str
7878             {
7879                 c { base }
7880                 l { base~west }
7881                 r { base~east }
7882                 j { base }
7883             }
7884         }
7885         { \box_use_drop:N \l_@@_cell_box } { } { }
7886     }
7887     \endpgfpicture
7888 }
7889 \group_end:
7890 }

```

For the command `\cellcolor` used within a sub-cell of a `\Block` (when the character `&` is used inside the cell).

```

7891 \cs_set_protected:Npn \@@_fill:nnnn #1 #2 #3 #4 #5
7892 {
7893     \pgfpicture
7894     \pgfrememberpicturepositiononpagetrue
7895     \pgf@relevantforpicturesizefalse
7896     \pgfpathrectanglecorners
7897     { \pgfpoint { #2 } { #3 } }
7898     { \pgfpoint { #4 } { #5 } }
7899     \pgfsetfillcolor { #1 }
7900     \pgfusepath { fill }
7901     \endpgfpicture
7902 }

```

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).

```

7903 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7904 {
7905     \group_begin:
7906     \tl_clear:N \l_@@_draw_tl
7907     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7908     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
7909     \pgfpicture
7910     \pgfrememberpicturepositiononpagetrue
7911     \pgf@relevantforpicturesizefalse
7912     \tl_if_empty:NF \l_@@_draw_tl
7913     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7914     \tl_if_eq:NNTF \l_@@_draw_tl \c_@@_default_tl
7915         { \CT@arc@ }
7916         { \@@_color:o \l_@@_draw_tl }
7917     }
7918     \pgfsetcornersarced
7919     {
7920         \pgfpoint
7921         { \l_@@_rounded_corners_dim }
7922         { \l_@@_rounded_corners_dim }
7923     }
7924     \@@_cut_on_hyphen:w #2 \q_stop
7925     \int_compare:nNnF \l_tmpa_tl > \c@iRow
7926     {
7927         \int_compare:nNnF \l_tmpb_tl > \c@jCol
7928         {
7929             \@@_qpoint:n { row - \l_tmpa_tl }
7930             \dim_set_eq:NN \l_tmpb_dim \pgf@y
7931             \@@_qpoint:n { col - \l_tmpb_tl }
7932             \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7933             \@@_cut_on_hyphen:w #3 \q_stop
7934             \int_compare:nNnT \l_tmpa_tl > \c@iRow
7935                 { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
7936             \int_compare:nNnT \l_tmpb_tl > \c@jCol
7937                 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
7938             \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7939             \dim_set_eq:NN \l_tmpa_dim \pgf@y
7940             \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7941             \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7942             \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7943             \pgfpathrectanglecorners
7944                 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7945                 { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7946             \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7947                 { \pgfusepathqstroke }
7948                 { \pgfusepath { stroke } }
7949         }
7950     }
7951     \endpgfpicture
7952     \group_end:
7953 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7954 \keys_define:nn { nicematrix / BlockStroke }
7955 {
7956     color .tl_set:N = \l_@@_draw_tl ,
7957     draw .code:n =
7958         \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7959     draw .default:n = default ,
7960     line-width .dim_set:N = \l_@@_line_width_dim ,
7961     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7962     rounded-corners .default:n = 4 pt
7963 }

```

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).

```

7964 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7965 {
7966     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7967     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
7968     \@@_cut_on_hyphen:w #2 \q_stop

```



```

7969 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7970 \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7971 \@@_cut_on_hyphen:w #3 \q_stop
7972 \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7973 \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7974 \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7975 {
7976   \use:e
7977   {
7978     \@@_vline:n
7979     {
7980       position = ##1 ,
7981       start = \l_@@_tmpc_tl ,
7982       end = \int_eval:n { \l_tmpa_tl - 1 } ,
7983       total-width = \dim_use:N \l_@@_line_width_dim
7984     }
7985   }
7986 }
7987 }
7988 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7989 {
7990   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7991   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
7992   \@@_cut_on_hyphen:w #2 \q_stop
7993   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7994   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7995   \@@_cut_on_hyphen:w #3 \q_stop
7996   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7997   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7998   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7999   {
8000     \use:e
8001     {
8002       \@@_hline:n
8003       {
8004         position = ##1 ,
8005         start = \l_@@_tmpd_tl ,
8006         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8007         total-width = \dim_use:N \l_@@_line_width_dim
8008       }
8009     }
8010   }
8011 }

```

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).

```

8012 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8013 {
8014   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8015   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8016   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8017     { \@@_error:n { borders~forbidden } }
8018     {
8019       \tl_clear_new:N \l_@@_borders_tikz_tl
8020       \keys_set:no
8021         { nicematrix / OnlyForTikzInBorders }
8022       \l_@@_borders_clist
8023       \@@_cut_on_hyphen:w #2 \q_stop
8024       \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8025       \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8026       \@@_cut_on_hyphen:w #3 \q_stop
8027       \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }

```

```

8028     \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8029     \@@_stroke_borders_block_i:
8030   }
8031 }

8032 \hook_gput_code:nnn { begindocument } { . }
8033 {
8034   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8035   {
8036     \c_@@_pgfortikzpicture_tl
8037     \@@_stroke_borders_block_ii:
8038     \c_@@_endpgfortikzpicture_tl
8039   }
8040 }

8041 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8042 {
8043   \pgfrememberpicturepositiononpagetrue
8044   \pgf@relevantforpicturesizefalse
8045   \CT@arc@
8046   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8047   \clist_if_in:NnT \l_@@_borders_clist { right }
8048   { \@@_stroke_vertical:n \l_tmpb_tl }
8049   \clist_if_in:NnT \l_@@_borders_clist { left }
8050   { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8051   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8052   { \@@_stroke_horizontal:n \l_tmpa_tl }
8053   \clist_if_in:NnT \l_@@_borders_clist { top }
8054   { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8055 }

8056 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8057 {
8058   tikz .code:n =
8059     \cs_if_exist:NTF \tikzpicture
8060     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8061     { \@@_error:n { tikz~in~borders~without~tikz } } ,
8062   tikz .value_required:n = true ,
8063   top .code:n = ,
8064   bottom .code:n = ,
8065   left .code:n = ,
8066   right .code:n = ,
8067   unknown .code:n = \@@_error:n { bad~border }
8068 }

```

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).

```

8069 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8070 {
8071   \@@_qpoint:n \l_@@_tmpc_tl
8072   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8073   \@@_qpoint:n \l_tmpa_tl
8074   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8075   \@@_qpoint:n { #1 }
8076   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8077   {
8078     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8079     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8080     \pgfusepathqstroke
8081   }
8082   {
8083     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8084     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8085   }
8086 }

```

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).

```

8087 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8088 {
8089   \@@_qpoint:n \l_@@_tmpd_tl
8090   \clist_if_in:NnTF \l_@@_borders_clist { left }
8091     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8092     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8093   \@@_qpoint:n \l_tmpb_tl
8094   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8095   \@@_qpoint:n { #1 }
8096   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8097     {
8098       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8099       \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8100       \pgfusepathqstroke
8101     }
8102     {
8103       \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8104         ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8105     }
8106 }

```

Here is the set of keys for the command \@@\_stroke\_borders\_block:nnn.

```

8107 \keys_define:nn { nicematrix / BlockBorders }
8108 {
8109   borders .clist_set:N = \l_@@_borders_clist ,
8110   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8111   rounded-corners .default:n = 4 pt ,
8112   line-width .dim_set:N = \l_@@_line_width_dim
8113 }

```

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.

```

8114 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }
8115 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8116 {
8117   \begin { tikzpicture }
8118   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because #5 is a list of lists.

```

8119   \clist_map_inline:nn { #1 }
8120   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8121     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8122     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8123     (
8124       [
8125         xshift = \dim_use:N \l_@@_offset_dim ,
8126         yshift = - \dim_use:N \l_@@_offset_dim
8127       ]
8128       #2 -| #3
8129     )
8130     rectangle
8131     (
8132     [

```

```

8133         xshift = - \dim_use:N \l_@@_offset_dim ,
8134         yshift = \dim_use:N \l_@@_offset_dim
8135     ]
8136     \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8137 ) ;
8138 }
8139 \end { tikzpicture }
8140 }

8141 \keys_define:nn { nicematrix / SpecialOffset }
8142 { 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.

```

8143 \cs_new_protected:Npn \@@_NullBlock:
8144 { \@@_collect_options:n { \@@_NullBlock_i: } }
8145 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8146 { }

```

## 27 How to draw the dotted lines transparently

```

8147 \cs_set_protected:Npn \@@_renew_matrix:
8148 {
8149   \RenewDocumentEnvironment { pmatrix } { }
8150   { \pNiceMatrix }
8151   { \endpNiceMatrix }
8152   \RenewDocumentEnvironment { vmatrix } { }
8153   { \vNiceMatrix }
8154   { \endvNiceMatrix }
8155   \RenewDocumentEnvironment { Vmatrix } { }
8156   { \VNiceMatrix }
8157   { \endVNiceMatrix }
8158   \RenewDocumentEnvironment { bmatrix } { }
8159   { \bNiceMatrix }
8160   { \endbNiceMatrix }
8161   \RenewDocumentEnvironment { Bmatrix } { }
8162   { \BNiceMatrix }
8163   { \endBNiceMatrix }
8164 }

```

## 28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8165 \keys_define:nn { nicematrix / Auto }
8166 {
8167   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8168   columns-type .value_required:n = true ,
8169   l .meta:n = { columns-type = l } ,
8170   r .meta:n = { columns-type = r } ,
8171   c .meta:n = { columns-type = c } ,
8172   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8173   delimiters / color .value_required:n = true ,
8174   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8175   delimiters / max-width .default:n = true ,
8176   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8177   delimiters .value_required:n = true ,

```

```

8178     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8179     rounded-corners .default:n = 4 pt
8180 }

8181 \NewDocumentCommand \AutoNiceMatrixWithDelims
8182 { m m O { } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8183 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8184 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8185 {

```

The group is for the protection of the keys.

```

8186     \group_begin:
8187     \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8188     \use:e
8189     {
8190         \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8191         { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8192         [ \exp_not:o \l_tmpa_tl ]
8193     }
8194     \int_if_zero:nT \l_@@_first_row_int
8195     {
8196         \int_if_zero:nT \l_@@_first_col_int { & }
8197         \prg_replicate:nn { #4 - 1 } { & }
8198         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8199     }
8200     \prg_replicate:mn { #3 }
8201     {
8202         \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`).

```

8203         \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8204         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8205     }
8206     \int_compare:nNnT \l_@@_last_row_int > { -2 }
8207     {
8208         \int_if_zero:nT \l_@@_first_col_int { & }
8209         \prg_replicate:nn { #4 - 1 } { & }
8210         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8211     }
8212     \end { NiceArrayWithDelims }
8213     \group_end:
8214 }

8215 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8216 {
8217     \cs_set_protected:cpn { #1 AutoNiceMatrix }
8218     {
8219         \bool_gset_true:N \g_@@_delims_bool
8220         \str_gset:Ne \g_@@_name_env_str { #1 AutoNiceMatrix }
8221         \AutoNiceMatrixWithDelims { #2 } { #3 }
8222     }
8223 }

8224 \@@_define_com:nnn p ( )
8225 \@@_define_com:nnn b [ ]
8226 \@@_define_com:nnn v | |
8227 \@@_define_com:nnn V \! \!
8228 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8229 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8230 {

```

```

8231 \group_begin:
8232 \bool_gset_false:N \g_@@_delims_bool
8233 \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8234 \group_end:
8235 }

```

## 29 The redefinition of the command `\dotfill`

```

8236 \cs_set_eq:NN \@@_old_dotfill \dotfill
8237 \cs_new_protected:Npn \@@_dotfill:
8238 {

```

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}`).

```

8239 \@@_old_dotfill
8240 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8241 }

```

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`.

```

8242 \cs_new_protected:Npn \@@_dotfill_i:
8243 { \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim \@@_old_dotfill }

```

## 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.

```

8244 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8245 {
8246 \tl_gput_right:Ne \g_@@_pre_code_after_tl
8247 {
8248 \@@_actually_diagbox:nnnnnn
8249 { \int_use:N \c@iRow }
8250 { \int_use:N \c@jCol }
8251 { \int_use:N \c@iRow }
8252 { \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.

```

8253 { \g_@@_row_style_tl \exp_not:n { #1 } }
8254 { \g_@@_row_style_tl \exp_not:n { #2 } }
8255 }

```

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`.

```

8256 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8257 {
8258 { \int_use:N \c@iRow }
8259 { \int_use:N \c@jCol }
8260 { \int_use:N \c@iRow }
8261 { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8262     { }
8263   }
8264 }

```

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.

```

8265 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8266 {
8267   \pgfpicture
8268   \pgf@relevantforpicturesizefalse
8269   \pgfrememberpicturepositiononpagetrue
8270   \@@_qpoint:n { row - #1 }
8271   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8272   \@@_qpoint:n { col - #2 }
8273   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8274   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8275   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8276   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8277   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8278   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8279   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8280 }

```

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.

```

8281   \CT@arc@
8282   \pgfsetroundcap
8283   \pgfusepathqstroke
8284 }
8285 \pgfset { inner~sep = 1 pt }
8286 \pgfscope
8287 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8288 \pgfnode { rectangle } { south~west }
8289 {
8290   \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8291   \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8292   \end { minipage }
8293 }
8294 { }
8295 { }
8296 \endpgfscope
8297 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8298 \pgfnode { rectangle } { north~east }
8299 {
8300   \begin { minipage } { 20 cm }
8301   \raggedleft
8302   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8303   \end { minipage }
8304 }
8305 { }
8306 { }
8307 \endpgfpicture
8308 }

```

## 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. 82.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
8309 \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 `\`.

```
8310 \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`.

```
8311 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8312 {
8313   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8314   \@@_CodeAfter_iv:n
8315 }
```

We catch the argument of the command `\end` (in `#1`).

```
8316 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8317 {
```

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.

```
8318   \str_if_eq:eeTF \currenvir { #1 }
8319   { \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`.

```
8320   {
8321     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8322     \@@_CodeAfter_ii:n
8323   }
8324 }
```

## 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.

```
8325 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8326 {
8327   \pgfpicture
8328   \pgfrememberpicturepositiononpagetrue
8329   \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.

```

8330 \@@_qpoint:n { row - 1 }
8331 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8332 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8333 \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).

```

8334 \bool_if:nTF { #3 }
8335 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8336 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8337 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8338 {
8339   \cs_if_exist:cT
8340   { \pgf @ sh @ ns @ \@@_env: - #1 - #2 }
8341   {
8342     \pgfpointanchor
8343     { \@@_env: - #1 - #2 }
8344     { \bool_if:nTF { #3 } { west } { east } }
8345     \dim_set:Nn \l_tmpa_dim
8346     { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8347   }
8348 }

```

Now we can put the delimiter with a node of PGF.

```

8349 \pgfset { inner~sep = \c_zero_dim }
8350 \dim_zero:N \nulldelimiterspace
8351 \pgftransformshift
8352 {
8353   \pgfpoint
8354   { \l_tmpa_dim }
8355   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8356 }
8357 \pgfnode
8358 { rectangle }
8359 { \bool_if:nTF { #3 } { east } { west } }
8360 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8361 \nullfont
8362 \c_math_toggle_token
8363 \@@_color:o \l_@@_delimiters_color_tl
8364 \bool_if:nTF { #3 } { \left #1 } { \left . }
8365 \vcenter
8366 {
8367   \nullfont
8368   \hrule \@height
8369   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8370   \@depth \c_zero_dim
8371   \@width \c_zero_dim
8372 }
8373 \bool_if:nTF { #3 } { \right . } { \right #1 }
8374 \c_math_toggle_token
8375 }
8376 { }
8377 { }
8378 \endpgfpicture
8379 }

```

### 33 The command `\SubMatrix`

```

8380 \keys_define:nn { nicematrix / sub-matrix }
8381 {
8382   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8383   extra-height .value_required:n = true ,
8384   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8385   left-xshift .value_required:n = true ,
8386   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8387   right-xshift .value_required:n = true ,
8388   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8389   xshift .value_required:n = true ,
8390   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8391   delimiters / color .value_required:n = true ,
8392   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8393   slim .default:n = true ,
8394   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8395   hlines .default:n = all ,
8396   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8397   vlines .default:n = all ,
8398   hvlines .meta:n = { hlines, vlines } ,
8399   hvlines .value_forbidden:n = true
8400 }
8401 \keys_define:nn { nicematrix }
8402 {
8403   SubMatrix .inherit:n = nicematrix / sub-matrix ,
8404   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8405   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8406   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8407 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8408 \keys_define:nn { nicematrix / SubMatrix }
8409 {
8410   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8411   delimiters / color .value_required:n = true ,
8412   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8413   hlines .default:n = all ,
8414   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8415   vlines .default:n = all ,
8416   hvlines .meta:n = { hlines, vlines } ,
8417   hvlines .value_forbidden:n = true ,
8418   name .code:n =
8419     \tl_if_empty:nTF { #1 }
8420     { \@@_error:n { Invalid-name } }
8421     {
8422       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8423       {
8424         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8425         { \@@_error:nn { Duplicate-name~for~SubMatrix } { #1 } }
8426         {
8427           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8428           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8429         }
8430       }
8431       { \@@_error:n { Invalid-name } }
8432     } ,
8433   name .value_required:n = true ,
8434   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8435   rules .value_required:n = true ,
8436   code .tl_set:N = \l_@@_code_tl ,

```

```

8437     code .value_required:n = true ,
8438     unknown .code:n = \@_error:n { Unknown~key~for~SubMatrix }
8439 }

8440 \NewDocumentCommand \@_SubMatrix_in_code_before { m m m m ! 0 { } }
8441 {
8442   \peek_remove_spaces:n
8443   {
8444     \tl_gput_right:Ne \g_@@_pre_code_after_tl
8445     {
8446       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8447       [
8448         delimiters / color = \l_@@_delimiters_color_tl ,
8449         hlines = \l_@@_submatrix_hlines_clist ,
8450         vlines = \l_@@_submatrix_vlines_clist ,
8451         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8452         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8453         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8454         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8455         #5
8456       ]
8457     }
8458     \@_SubMatrix_in_code_before_i { #2 } { #3 }
8459   }
8460 }

8461 \NewDocumentCommand \@_SubMatrix_in_code_before_i
8462 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8463 { \@_SubMatrix_in_code_before_i:n n n #1 #2 }

8464 \cs_new_protected:Npn \@_SubMatrix_in_code_before_i:n n n #1 #2 #3 #4
8465 {
8466   \seq_gput_right:Ne \g_@@_submatrix_seq
8467   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8468     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8469     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8470     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8471     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8472   }
8473 }

```

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`.

```

8474 \hook_gput_code:nnn { begindocument } { . }
8475 {
8476   \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8477   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

```

8478 \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8479 {
8480   \peek_remove_spaces:n
8481   {
8482     \@@_sub_matrix:nnnnnnn
8483     { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8484   }
8485 }
8486 }

```

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).

```

8487 \NewDocumentCommand \@@_compute_i_j:nn
8488 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8489 { \@@_compute_i_j:nnnn #1 #2 }

8490 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8491 {
8492   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8493   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8494   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8495   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8496   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8497     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8498   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8499     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8500   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8501     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8502   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8503     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8504 }

8505 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8506 {
8507   \group_begin:

```

The four following token lists correspond to the position of the  $\backslash$ SubMatrix.

```

8508 \@@_compute_i_j:nn { #2 } { #3 }
8509 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8510   { \cs_set_nopar:Npn \arraystretch { 1 } }
8511 \bool_lazy_or:nnTF
8512   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8513   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8514   { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8515 {
8516   \str_clear_new:N \l_@@_submatrix_name_str
8517   \keys_set:nn { nicematrix / SubMatrix } { #5 }
8518   \pgfpicture
8519   \pgfrememberpicturepositiononpagetrue
8520   \pgf@relevantforpicturesizefalse
8521   \pgfset { inner~sep = \c_zero_dim }
8522   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8523   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of  $\backslash$ int\_step\_inline:nnn is provided by curriification.

```

8524 \bool_if:NTF \l_@@_submatrix_slim_bool
8525   { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8526   { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8527   {
8528     \cs_if_exist:cT
8529       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8530     {
8531       \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8532       \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim

```

```

8533         { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8534     }
8535     \cs_if_exist:cT
8536     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8537     {
8538         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8539         \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8540         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8541     }
8542 }
8543 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8544 { \@@_error:nn { Impossible-delimiter } { left } }
8545 {
8546     \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8547     { \@@_error:nn { Impossible-delimiter } { right } }
8548     { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8549 }
8550 \endpgfpicture
8551 }
8552 \group_end:
8553 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8554 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8555 {
8556     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8557     \dim_set:Nn \l_@@_y_initial_dim
8558     {
8559         \fp_to_dim:n
8560         {
8561             \pgf@y
8562             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8563         }
8564     }
8565     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8566     \dim_set:Nn \l_@@_y_final_dim
8567     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8568     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8569     {
8570         \cs_if_exist:cT
8571         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8572         {
8573             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8574             \dim_set:Nn \l_@@_y_initial_dim
8575             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8576         }
8577         \cs_if_exist:cT
8578         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8579         {
8580             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8581             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
8582             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8583         }
8584     }
8585     \dim_set:Nn \l_tmpa_dim
8586     {
8587         \l_@@_y_initial_dim - \l_@@_y_final_dim +
8588         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8589     }
8590     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8591 \group_begin:
8592 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8593 \@@_set_CT@arc@:o \l_@@_rules_color_tl
8594 \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`.

```

8595 \seq_map_inline:Nn \g_@@_cols_vlism_seq
8596 {
8597   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8598   {
8599     \int_compare:nNnT
8600       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8601     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8602       \@@_qpoint:n { col - ##1 }
8603       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8604       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8605       \pgfusepathqstroke
8606     }
8607   }
8608 }

```

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.

```

8609 \tl_if_eq:NNTF \l_@@_submatrix_vlines_clist \c_@@_all_tl
8610 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8611 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8612 {
8613   \bool_lazy_and:nnTF
8614     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8615     {
8616       \int_compare_p:nNn
8617         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8618     {
8619       \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8620       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8621       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8622       \pgfusepathqstroke
8623     }
8624     { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8625 }

```

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.

```

8626 \tl_if_eq:NNTF \l_@@_submatrix_hlines_clist \c_@@_all_tl
8627 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8628 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8629 {
8630   \bool_lazy_and:nnTF
8631     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8632     {
8633       \int_compare_p:nNn
8634         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8635     {
8636       \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8637 \group_begin:

```

We compute in `\l_tmpa_dim` the  $x$ -value of the left end of the rule.

```

8638 \dim_set:Nn \l_tmpa_dim

```

```

8639         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8640     \str_case:nn { #1 }
8641     {
8642         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8643         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8644         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8645         }
8646     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the  $x$ -value of the right end of the rule.

```

8647     \dim_set:Nn \l_tmpb_dim
8648     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8649     \str_case:nn { #2 }
8650     {
8651         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8652         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8653         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8654     }
8655     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8656     \pgfusepathqstroke
8657     \group_end:
8658 }
8659 { \@@_error:nnn { Wrong-line-in-SubMatrix } { horizontal } { ##1 } }
8660 }

```

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).

```

8661     \str_if_empty:NF \l_@@_submatrix_name_str
8662     {
8663         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8664         \l_@@_x_initial_dim \l_@@_y_initial_dim
8665         \l_@@_x_final_dim \l_@@_y_final_dim
8666     }
8667     \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`.

```

8668     \begin { pgfscope }
8669     \pgftransformshift
8670     {
8671         \pgfpoint
8672         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8673         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8674     }
8675     \str_if_empty:NTF \l_@@_submatrix_name_str
8676     { \@@_node_left:nn #1 { } }
8677     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8678     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8679     \pgftransformshift
8680     {
8681         \pgfpoint
8682         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8683         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8684     }
8685     \str_if_empty:NTF \l_@@_submatrix_name_str
8686     { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8687     {
8688         \@@_node_right:nnnn #2
8689         { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8690     }

```

```

8691 \cs_set_eq:NN \pgfpointanchor \@_pgfpointanchor:n
8692 \flag_clear_new:N \l_@@_code_flag
8693 \l_@@_code_tl
8694 }

```

In the key code of the command `\SubMatrix` there may be Tikz instructions. We want that, in these instructions, the  $i$  and  $j$  in specifications of nodes of the forms  $i-j$ ,  $row-i$ ,  $col-j$  and  $i-|j$  refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

8695 \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.

```

8696 \cs_new_protected:Npn \@_pgfpointanchor:n #1
8697 {
8698   \use:e
8699   { \exp_not:N \@_old_pgfpointanchor { \@_pgfpointanchor_i:nn #1 } }
8700 }

```

In fact, the argument of `\pgfpointanchor` is always of the form `\a_command { name_of_node }` where “name\_of\_node” is the name of the Tikz node without the potential prefix and suffix. That's why we catch two arguments and work only on the second by trying (first) to extract an hyphen -.

```

8701 \cs_new:Npn \@_pgfpointanchor_i:nn #1 #2
8702 { #1 { \@_pgfpointanchor_ii:w #2 - \q_stop } }

```

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.

```

8703 \tl_const:Nn \c_@@_integers_alist_tl
8704 {
8705   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8706   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8707   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8708   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8709 }

```

```

8710 \cs_new:Npn \@_pgfpointanchor_ii:w #1-#2\q_stop
8711 {

```

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  $i-|j$ . 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`.

```

8712 \tl_if_empty:nTF { #2 }
8713 {
8714   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8715   {
8716     \flag_raise:N \l_@@_code_flag
8717     \int_if_even:nTF { \flag_height:N \l_@@_code_if_flag }
8718     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8719     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8720   }
8721   { #1 }
8722 }

```



If there is an hyphen, we have to see whether we have a node of the form  $i-j$ ,  $row-i$  or  $col-j$ .

```
8723     { \@@_pgfpointanchor_iii:w { #1 } #2 }
8724   }
```

There was an hyphen in the name of the node and that's why we have to retrieve the extra hyphen we have put (cf. `\@@_pgfpointanchor_i:nn`).

```
8725 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8726   {
8727     \str_case:nnF { #1 }
8728     {
8729       { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8730       { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8731     }
```

Now the case of a node of the form  $i-j$ .

```
8732     {
8733       \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8734       - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8735     }
8736   }
```

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`).

```
8737 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8738   {
8739     \pgfnode
8740     { rectangle }
8741     { east }
8742     {
8743       \nullfont
8744       \c_math_toggle_token
8745       \@@_color:o \l_@@_delimiters_color_tl
8746       \left #1
8747       \vcenter
8748       {
8749         \nullfont
8750         \hrule \@height \l_tmpa_dim
8751         \@depth \c_zero_dim
8752         \@width \c_zero_dim
8753       }
8754       \right .
8755       \c_math_toggle_token
8756     }
8757     { #2 }
8758     { }
8759   }
```

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.

```
8760 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8761   {
8762     \pgfnode
8763     { rectangle }
8764     { west }
8765     {
8766       \nullfont
8767       \c_math_toggle_token
8768       \colorlet { current-color } { . }
8769       \@@_color:o \l_@@_delimiters_color_tl
8770       \left .
```

```

8771     \vcenter
8772     {
8773         \nullfont
8774         \hrule \@height \l_tmpa_dim
8775             \@depth \c_zero_dim
8776             \@width \c_zero_dim
8777     }
8778     \right #1
8779     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8780     ~ { \color { current-color } \smash { #4 } }
8781     \c_math_toggle_token
8782 }
8783 { #2 }
8784 { }
8785 }

```

### 34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8786 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8787 {
8788     \peek_remove_spaces:n
8789     { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8790 }
8791 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8792 {
8793     \peek_remove_spaces:n
8794     { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8795 }
8796 \keys_define:nn { nicematrix / Brace }
8797 {
8798     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8799     left-shorten .default:n = true ,
8800     left-shorten .value_forbidden:n = true ,
8801     right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8802     right-shorten .default:n = true ,
8803     right-shorten .value_forbidden:n = true ,
8804     shorten .meta:n = { left-shorten , right-shorten } ,
8805     shorten .value_forbidden:n = true ,
8806     yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8807     yshift .value_required:n = true ,
8808     yshift .initial:n = \c_zero_dim ,
8809     color .tl_set:N = \l_tmpa_tl ,
8810     color .value_required:n = true ,
8811     unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8812 }

```

`#1` is the first cell of the rectangle (with the syntax `i-lj`; `#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`.

```

8813 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8814 {
8815     \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8816 \@@_compute_i_j:nn { #1 } { #2 }
8817 \bool_lazy_or:nnTF

```

```

8818 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8819 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8820 {
8821   \str_if_eq:eeTF { #5 } { under }
8822   { \@@_error:nn { Construct-too-large } { \UnderBrace } }
8823   { \@@_error:nn { Construct-too-large } { \OverBrace } }
8824 }
8825 {
8826   \tl_clear:N \l_tmpa_tl
8827   \keys_set:nn { nicematrix / Brace } { #4 }
8828   \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8829   \pgfpicture
8830   \pgfrememberpicturepositiononpagetrue
8831   \pgf@relevantforpicturesizefalse
8832   \bool_if:NT \l_@@_brace_left_shorten_bool
8833   {
8834     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8835     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8836     {
8837       \cs_if_exist:cT
8838       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8839       {
8840         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8841
8842         \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8843         { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8844       }
8845     }
8846   }
8847   \bool_lazy_or:nnT
8848   { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8849   { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8850   {
8851     \@@_qpoint:n { col - \l_@@_first_j_tl }
8852     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8853   }
8854   \bool_if:NT \l_@@_brace_right_shorten_bool
8855   {
8856     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8857     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8858     {
8859       \cs_if_exist:cT
8860       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8861       {
8862         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8863         \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8864         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8865       }
8866     }
8867   }
8868   \bool_lazy_or:nnT
8869   { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8870   { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8871   {
8872     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8873     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8874   }
8875   \pgfset { inner~sep = \c_zero_dim }
8876   \str_if_eq:eeTF { #5 } { under }
8877   { \@@_underbrace_i:n { #3 } }
8878   { \@@_overbrace_i:n { #3 } }
8879   \endpgfpicture
8880 }

```

```

8881 \group_end:
8882 }

```

The argument is the text to put above the brace.

```

8883 \cs_new_protected:Npn \@@_overbrace_i:n #1
8884 {
8885   \@@_qpoint:n { row - \l_@@_first_i_tl }
8886   \pgftransformshift
8887   {
8888     \pgfpoint
8889     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8890     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8891   }
8892   \pgfnode
8893   { rectangle }
8894   { south }
8895   {
8896     \vtop
8897     {
8898       \group_begin:
8899       \everycr { }
8900       \halign
8901       {
8902         \hfil ## \hfil \crcr
8903         \bool_if:NTF \l_@@_tabular_bool
8904         { \begin { tabular } { c } #1 \end { tabular } }
8905         { $ \begin { array } { c } #1 \end { array } $ }
8906         \cr
8907         \c_math_toggle_token
8908         \overbrace
8909         {
8910           \hbox_to_wd:nn
8911           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8912           { }
8913         }
8914         \c_math_toggle_token
8915         \cr
8916       }
8917     }
8918   }
8919 }
8920 { }
8921 { }
8922 }

```

The argument is the text to put under the brace.

```

8923 \cs_new_protected:Npn \@@_underbrace_i:n #1
8924 {
8925   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8926   \pgftransformshift
8927   {
8928     \pgfpoint
8929     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8930     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8931   }
8932   \pgfnode
8933   { rectangle }
8934   { north }
8935   {
8936     \group_begin:
8937     \everycr { }
8938     \vbox
8939     {

```

```

8940     \halign
8941     {
8942     \hfil ## \hfil \crcr
8943     \c_math_toggle_token
8944     \underbrace
8945     {
8946     \hbox_to_wd:nn
8947     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8948     { }
8949     }
8950     \c_math_toggle_token
8951     \cr
8952     \bool_if:NTF \l_@@_tabular_bool
8953     { \begin { tabular } { c } #1 \end { tabular } }
8954     { $ \begin { array } { c } #1 \end { array } $ }
8955     \cr
8956     }
8957   }
8958   \group_end:
8959 }
8960 { }
8961 { }
8962 }

```

## 35 The command TikzEveryCell

```

8963 \bool_new:N \l_@@_not_empty_bool
8964 \bool_new:N \l_@@_empty_bool
8965
8966 \keys_define:nn { nicematrix / TikzEveryCell }
8967 {
8968   not-empty .code:n =
8969     \bool_lazy_or:nnTF
8970     \l_@@_in_code_after_bool
8971     \g_@@_recreate_cell_nodes_bool
8972     { \bool_set_true:N \l_@@_not_empty_bool }
8973     { \@@_error:n { detection~of~empty~cells } } } ,
8974   not-empty .value_forbidden:n = true ,
8975   empty .code:n =
8976     \bool_lazy_or:nnTF
8977     \l_@@_in_code_after_bool
8978     \g_@@_recreate_cell_nodes_bool
8979     { \bool_set_true:N \l_@@_empty_bool }
8980     { \@@_error:n { detection~of~empty~cells } } } ,
8981   empty .value_forbidden:n = true ,
8982   unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
8983 }
8984
8985
8986 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
8987 {
8988   \IfPackageLoadedTF { tikz }
8989   {
8990     \group_begin:
8991     \keys_set:nn { nicematrix / TikzEveryCell } { #1 }

```

The inner pair of braces in the following line is mandatory because, the last argument of `\@@_tikz:nnnn` is a *list of lists* of TikZ keys.

```

8992     \tl_set:Nn \l_tmpa_tl { { #2 } }
8993     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq

```

```

8994     { \@@_for_a_block:nnnnn ##1 }
8995     \@@_all_the_cells:
8996     \group_end:
8997   }
8998   { \@@_error:n { TikzEveryCell~without~tikz } }
8999 }
9000
9001 \tl_new:N \@@_i_tl
9002 \tl_new:N \@@_j_tl
9003
9004
9005 \cs_new_protected:Nn \@@_all_the_cells:
9006 {
9007   \int_step_variable:nNn \c@iRow \@@_i_tl
9008   {
9009     \int_step_variable:nNn \c@jCol \@@_j_tl
9010     {
9011       \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9012       {
9013         \clist_if_in:NeF \l_@@_corners_cells_clist
9014         { \@@_i_tl - \@@_j_tl }
9015         {
9016           \bool_set_false:N \l_tmpa_bool
9017           \cs_if_exist:cTF
9018           { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9019           {
9020             \bool_if:NF \l_@@_empty_bool
9021             { \bool_set_true:N \l_tmpa_bool }
9022           }
9023           {
9024             \bool_if:NF \l_@@_not_empty_bool
9025             { \bool_set_true:N \l_tmpa_bool }
9026           }
9027           \bool_if:NT \l_tmpa_bool
9028           {
9029             \@@_block_tikz:onnnn
9030             \l_tmpa_tl \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl
9031           }
9032         }
9033       }
9034     }
9035   }
9036 }
9037
9038 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9039 {
9040   \bool_if:NF \l_@@_empty_bool
9041   {
9042     \@@_block_tikz:onnnn
9043     \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9044   }
9045   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9046 }
9047
9048 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9049 {
9050   \int_step_inline:nnn { #1 } { #3 }
9051   {
9052     \int_step_inline:nnn { #2 } { #4 }
9053     { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9054   }
9055 }

```

## 36 The command \ShowCellNames

```

9056 \NewDocumentCommand \@@_ShowCellNames { }
9057 {
9058   \bool_if:NT \l_@@_in_code_after_bool
9059   {
9060     \pgfpicture
9061     \pgfrememberpicturepositiononpagetrue
9062     \pgf@relevantforpicturesizefalse
9063     \pgfpathrectanglecorners
9064     { \@@_qpoint:n { 1 } }
9065     {
9066       \@@_qpoint:n
9067       { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9068     }
9069     \pgfsetfillopacity { 0.75 }
9070     \pgfsetfillcolor { white }
9071     \pgfusepathqfill
9072     \endpgfpicture
9073   }
9074   \dim_gzero_new:N \g_@@_tmpc_dim
9075   \dim_gzero_new:N \g_@@_tmpd_dim
9076   \dim_gzero_new:N \g_@@_tmpe_dim
9077   \int_step_inline:nn \c@iRow
9078   {
9079     \bool_if:NTF \l_@@_in_code_after_bool
9080     {
9081       \pgfpicture
9082       \pgfrememberpicturepositiononpagetrue
9083       \pgf@relevantforpicturesizefalse
9084     }
9085     { \begin { pgfpicture } }
9086     \@@_qpoint:n { row - ##1 }
9087     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9088     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9089     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9090     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9091     \bool_if:NTF \l_@@_in_code_after_bool
9092     { \endpgfpicture }
9093     { \end { pgfpicture } }
9094     \int_step_inline:nn \c@jCol
9095     {
9096       \hbox_set:Nn \l_tmpa_box
9097       {
9098         \normalfont \Large \sffamily \bfseries
9099         \bool_if:NTF \l_@@_in_code_after_bool
9100         { \color { red } }
9101         { \color { red ! 50 } }
9102         ##1 - #####1
9103       }
9104       \bool_if:NTF \l_@@_in_code_after_bool
9105       {
9106         \pgfpicture
9107         \pgfrememberpicturepositiononpagetrue
9108         \pgf@relevantforpicturesizefalse
9109       }
9110       { \begin { pgfpicture } }
9111       \@@_qpoint:n { col - #####1 }
9112       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9113       \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
9114       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9115       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x

```

```

9116     \bool_if:NTF \l_@@_in_code_after_bool
9117     { \endpgfpicture }
9118     { \end { pgfpicture } }
9119     \fp_set:Nn \l_tmpa_fp
9120     {
9121         \fp_min:nn
9122         {
9123             \fp_min:nn
9124             { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9125             { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9126         }
9127         { 1.0 }
9128     }
9129     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9130     \pgfpicture
9131     \pgfrememberpicturerepositiononpagetrue
9132     \pgf@relevantforpicturesizefalse
9133     \pgftransformshift
9134     {
9135         \pgfpoint
9136         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9137         { \dim_use:N \g_tmpa_dim }
9138     }
9139     \pgfnode
9140     { rectangle }
9141     { center }
9142     { \box_use:N \l_tmpa_box }
9143     { }
9144     { }
9145     \endpgfpicture
9146 }
9147 }
9148 }

```

## 37 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.

```

9149 \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.

```

9150 \bool_new:N \g_@@_footnote_bool
9151 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9152 {
9153     The~key~'\l_keys_key_str'~is~unknown. \\
9154     That~key~will~be~ignored. \\
9155     For~a~list~of~the~available~keys,~type~H~<return>.
9156 }
9157 {
9158     The~available~keys~are~(in~alphabetic~order):~
9159     footnote,~
9160     footnotehyper,~
9161     messages~for~Overleaf,~
9162     renew~dots,~and~
9163     renew~matrix.

```



```

9164 }
9165 \keys_define:nn { nicematrix / Package }
9166 {
9167   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9168   renew-dots .value_forbidden:n = true ,
9169   renew-matrix .code:n = \@@_renew_matrix: ,
9170   renew-matrix .value_forbidden:n = true ,
9171   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9172   footnote .bool_set:N = \g_@@_footnote_bool ,
9173   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,

```

The test for a potential modification of array has been deleted. You keep the following key only for compatibility but maybe we will delete it.

```

9174   no-test-for-array .code:n = \prg_do_nothing: ,
9175   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9176 }
9177 \ProcessKeysOptions { nicematrix / Package }

```

```

9178 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9179 {
9180   You-can't-use-the-option-'footnote'~because~the~package~
9181   footnotehyper~has~already~been~loaded.~
9182   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9183   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9184   of~the~package~footnotehyper.\\
9185   The~package~footnote~won't~be~loaded.
9186 }

```

```

9187 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9188 {
9189   You-can't-use-the-option-'footnotehyper'~because~the~package~
9190   footnote~has~already~been~loaded.~
9191   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9192   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9193   of~the~package~footnote.\\
9194   The~package~footnotehyper~won't~be~loaded.
9195 }

```

```

9196 \bool_if:NT \g_@@_footnote_bool
9197 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9198   \IfClassLoadedTF { beamer }
9199     { \bool_set_false:N \g_@@_footnote_bool }
9200   {
9201     \IfPackageLoadedTF { footnotehyper }
9202       { \@@_error:n { footnote~with~footnotehyper~package } }
9203       { \usepackage { footnote } }
9204   }
9205 }

```

```

9206 \bool_if:NT \g_@@_footnotehyper_bool
9207 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9208   \IfClassLoadedTF { beamer }
9209     { \bool_set_false:N \g_@@_footnote_bool }
9210   {
9211     \IfPackageLoadedTF { footnote }
9212       { \@@_error:n { footnotehyper~with~footnote~package } }
9213       { \usepackage { footnotehyper } }
9214   }

```

```

9215     \bool_set_true:N \g_@@_footnote_bool
9216   }

```

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}`.

## 38 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.

```

9217 \bool_new:N \l_@@_underscore_loaded_bool
9218 \IfPackageLoadedT { underscore }
9219   { \bool_set_true:N \l_@@_underscore_loaded_bool }
9220 \hook_gput_code:nnn { begindocument } { . }
9221 {
9222   \bool_if:NF \l_@@_underscore_loaded_bool
9223     {
9224       \IfPackageLoadedT { underscore }
9225         { \@@_error:n { underscore-after~nicematrix } }
9226     }
9227 }

```

## 39 Error messages of the package

```

9228 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9229   { \str_const:Nn \c_@@_available_keys_str { } }
9230   {
9231     \str_const:Nn \c_@@_available_keys_str
9232       { For~a~list~of~the~available~keys,~type-H~<return>. }
9233   }
9234 \seq_new:N \g_@@_types_of_matrix_seq
9235 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9236   {
9237     NiceMatrix ,
9238     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9239   }
9240 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9241   { \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`.

```

9242 \cs_new_protected:Npn \@@_error_too_much_cols:
9243   {
9244     \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9245       { \@@_fatal:nm { too-much-cols-for-array } }
9246     \int_compare:nNnT \l_@@_last_col_int = { -2 }
9247       { \@@_fatal:n { too-much-cols-for-matrix } }
9248     \int_compare:nNnT \l_@@_last_col_int = { -1 }
9249       { \@@_fatal:n { too-much-cols-for-matrix } }
9250     \bool_if:NF \l_@@_last_col_without_value_bool
9251       { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }

```

9252 }

The following command must *not* be protected since it's used in an error message.

```
9253 \cs_new:Npn \@@_message_hdotsfor:
9254 {
9255   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9256   { ~Maybe-your-use-of-\token_to_str:N \Hdotsfor\ is-incorrect.}
9257 }
9258 \@@_msg_new:nn { hvlines,~rounded-corners-and-corners }
9259 {
9260   Incompatible-options.\\
9261   You-should-not-use-'hvlines',~'rounded-corners'~and-'corners'~at-this-time.\\
9262   The-output-will-not-be-reliable.
9263 }
9264 \@@_msg_new:nn { negative-weight }
9265 {
9266   Negative-weight.\\
9267   The-weight-of-the-'X'-columns-must-be-positive-and-you-have-used-
9268   the-value-\int_use:N \l_@@_weight_int'.\\
9269   The-absolute-value-will-be-used.
9270 }
9271 \@@_msg_new:nn { last-col-not-used }
9272 {
9273   Column-not-used.\\
9274   The-key-'last-col'-is-in-force-but-you-have-not-used-that-last-column-
9275   in-your-\@@_full_name_env:.~However,~you-can-go-on.
9276 }
9277 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9278 {
9279   Too-much-columns.\\
9280   In-the-row~\int_eval:n { \c@iRow },~
9281   you-try-to-use-more-columns-
9282   than-allowed-by-your-\@@_full_name_env:. \@@_message_hdotsfor:\
9283   The-maximal-number-of-columns-is-\int_eval:n { \l_@@_last_col_int - 1 }~
9284   (plus-the-exterior-columns).~This-error-is-fatal.
9285 }
9286 \@@_msg_new:nn { too-much-cols-for-matrix }
9287 {
9288   Too-much-columns.\\
9289   In-the-row~\int_eval:n { \c@iRow },~
9290   you-try-to-use-more-columns-than-allowed-by-your~
9291   \@@_full_name_env:. \@@_message_hdotsfor:\ Recall-that-the-maximal-
9292   number-of-columns-for-a-matrix~(excepted-the-potential-exterior-
9293   columns)-is-fixed-by-the-LaTeX-counter-'MaxMatrixCols'.~
9294   Its-current-value-is-\int_use:N \c@MaxMatrixCols\ (use~
9295   \token_to_str:N \setcounter\ to-change-that-value).~
9296   This-error-is-fatal.
9297 }
9298 \@@_msg_new:nn { too-much-cols-for-array }
9299 {
9300   Too-much-columns.\\
9301   In-the-row~\int_eval:n { \c@iRow },~
9302   ~you-try-to-use-more-columns-than-allowed-by-your~
9303   \@@_full_name_env:. \@@_message_hdotsfor:\ The-maximal-number-of-columns-is-
9304   \int_use:N \g_@@_static_num_of_col_int\
9305   ~ (plus-the-potential-exterior-ones).~
9306   This-error-is-fatal.
9307 }
9308 \@@_msg_new:nn { columns-not-used }
9309 {
9310   Columns-not-used.\\
```

```

9311     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9312     \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\
9313     The~columns~you~did~not~used~won't~be~created.\\
9314     You~won't~have~similar~error~message~till~the~end~of~the~document.
9315 }
9316 \@@_msg_new:nn { empty~preamble }
9317 {
9318     Empty~preamble.\\
9319     The~preamble~of~your~\@@_full_name_env:\ is~empty.\\
9320     This~error~is~fatal.
9321 }
9322 \@@_msg_new:nn { in~first~col }
9323 {
9324     Erroneous~use.\\
9325     You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
9326     That~command~will~be~ignored.
9327 }
9328 \@@_msg_new:nn { in~last~col }
9329 {
9330     Erroneous~use.\\
9331     You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9332     That~command~will~be~ignored.
9333 }
9334 \@@_msg_new:nn { in~first~row }
9335 {
9336     Erroneous~use.\\
9337     You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9338     That~command~will~be~ignored.
9339 }
9340 \@@_msg_new:nn { in~last~row }
9341 {
9342     You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9343     That~command~will~be~ignored.
9344 }
9345 \@@_msg_new:nn { caption~outside~float }
9346 {
9347     Key~caption~forbidden.\\
9348     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9349     environment.~This~key~will~be~ignored.
9350 }
9351 \@@_msg_new:nn { short~caption~without~caption }
9352 {
9353     You~should~not~use~the~key~'short~caption'~without~'caption'.~
9354     However,~your~'short~caption'~will~be~used~as~'caption'.
9355 }
9356 \@@_msg_new:nn { double~closing~delimiter }
9357 {
9358     Double~delimiter.\\
9359     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9360     delimiter.~This~delimiter~will~be~ignored.
9361 }
9362 \@@_msg_new:nn { delimiter~after~opening }
9363 {
9364     Double~delimiter.\\
9365     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9366     delimiter.~That~delimiter~will~be~ignored.
9367 }
9368 \@@_msg_new:nn { bad~option~for~line~style }
9369 {

```

```

9370     Bad-line-style.\
9371     Since-you-haven't-loaded-Tikz,~the-only~value~you~can~give~to~'line-style'~
9372     is~'standard'.~That~key~will~be~ignored.
9373 }
9374 \@@_msg_new:nn { Identical-notes-in-caption }
9375 {
9376     Identical~tabular~notes.\
9377     You~can't~put~several~notes~with~the~same~content~in~
9378     \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\
9379     If~you~go~on,~the~output~will~probably~be~erroneous.
9380 }
9381 \@@_msg_new:nn { tabularnote~below~the~tabular }
9382 {
9383     \token_to_str:N \tabularnote\ forbidden\
9384     You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9385     of~your~tabular~because~the~caption~will~be~composed~below~
9386     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9387     key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\
9388     Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9389     no~similar~error~will~raised~in~this~document.
9390 }
9391 \@@_msg_new:nn { Unknown~key~for~rules }
9392 {
9393     Unknown~key.\
9394     There~is~only~two~keys~available~here:~width~and~color.\
9395     Your~key~'\l_keys_key_str'~will~be~ignored.
9396 }
9397 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9398 {
9399     Unknown~key.\
9400     There~is~only~two~keys~available~here:~
9401     'empty'~and~'not~empty'.\
9402     Your~key~'\l_keys_key_str'~will~be~ignored.
9403 }
9404 \@@_msg_new:nn { Unknown~key~for~rotate }
9405 {
9406     Unknown~key.\
9407     The~only~key~available~here~is~'c'.\
9408     Your~key~'\l_keys_key_str'~will~be~ignored.
9409 }
9410 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9411 {
9412     Unknown~key.\
9413     The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9414     It~you~go~on,~you~will~probably~have~other~errors. \
9415     \c_@@_available_keys_str
9416 }
9417 {
9418     The~available~keys~are~(in~alphabetic~order):~
9419     ccommand,~
9420     color,~
9421     command,~
9422     dotted,~
9423     letter,~
9424     multiplicity,~
9425     sep~color,~
9426     tikz,~and~total~width.
9427 }
9428 \@@_msg_new:nnn { Unknown~key~for~xdots }
9429 {
9430     Unknown~key.\

```

```

9431     The-key~'\l_keys_key_str'~is-unknown~for~a~command~for~drawing~dotted~rules.\\
9432     \c_@@_available_keys_str
9433   }
9434   {
9435     The-available-keys~are~(in~alphabetic~order):~
9436     'color',~
9437     'horizontal-labels',~
9438     'inter',~
9439     'line-style',~
9440     'radius',~
9441     'shorten',~
9442     'shorten-end'~and~'shorten-start'.
9443   }

9444 \@@_msg_new:nn { Unknown-key~for~rowcolors }
9445   {
9446     Unknown-key.\\
9447     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9448     (and~you~try~to~use~'\l_keys_key_str')\\
9449     That~key~will~be~ignored.
9450   }

9451 \@@_msg_new:nn { label~without~caption }
9452   {
9453     You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9454     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9455   }

9456 \@@_msg_new:nn { W-warning }
9457   {
9458     Line~\msg_line_number:.~The~cell~is~too~wide~for~your~column~'W'~
9459     (row~\int_use:N \c@iRow).
9460   }

9461 \@@_msg_new:nn { Construct-too-large }
9462   {
9463     Construct-too-large.\\
9464     Your~command~\token_to_str:N #1
9465     can't~be~drawn~because~your~matrix~is~too~small.\\
9466     That~command~will~be~ignored.
9467   }

9468 \@@_msg_new:nn { underscore~after~nicematrix }
9469   {
9470     Problem~with~'underscore'.\\
9471     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9472     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9473     '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9474   }

9475 \@@_msg_new:nn { ampersand~in~light-syntax }
9476   {
9477     Ampersand~forbidden.\\
9478     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9479     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9480   }

9481 \@@_msg_new:nn { double-backslash~in~light-syntax }
9482   {
9483     Double~backslash~forbidden.\\
9484     You~can't~use~\token_to_str:N
9485     \\~to~separate~rows~because~the~key~'light-syntax'~
9486     is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9487     (set~by~the~key~'end-of-row').~This~error~is~fatal.
9488   }

9489 \@@_msg_new:nn { hlines~with~color }
9490   {

```

```

9491     Incompatible-keys.\
9492     You-can't-use-the-keys-'hlines',~'vlines'~or~'hvlines'~for~a~
9493     '\token_to_str:N \Block'~when~the-key~'color'~or~'draw'~is~used.\
9494     However,~you-can~put~several~commands~\token_to_str:N \Block.\
9495     Your-key-will-be-discarded.
9496   }
9497   \@@_msg_new:nn { bad-value-for-baseline }
9498   {
9499     Bad-value-for-baseline.\
9500     The-value-given-to-'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9501     valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int\ and~
9502     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9503     the-form-'line-i'.\
9504     A-value-of~1~will-be-used.
9505   }
9506   \@@_msg_new:nn { detection-of-empty-cells }
9507   {
9508     Problem-with-'not-empty'\
9509     For-technical-reasons,~you-must-activate~
9510     'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9511     in-order-to-use-the-key~'\l_keys_key_str'.\
9512     That-key-will-be-ignored.
9513   }
9514   \@@_msg_new:nn { siunitx-not-loaded }
9515   {
9516     siunitx-not-loaded\
9517     You-can't-use-the-columns-'S'~because-'siunitx'~is~not~loaded.\
9518     That-error-is-fatal.
9519   }
9520   \@@_msg_new:nn { Invalid-name }
9521   {
9522     Invalid-name.\
9523     You-can't-give-the-name~'\l_keys_value_tl'~to~a~\token_to_str:N
9524     \SubMatrix\ of~your~\@@_full_name_env:.\
9525     A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\
9526     This-key-will-be-ignored.
9527   }
9528   \@@_msg_new:nn { Wrong-line-in-SubMatrix }
9529   {
9530     Wrong-line.\
9531     You-try-to-draw~a~#1~line~of~number~'#2'~in~a~
9532     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9533     number~is~not~valid.~It~will~be~ignored.
9534   }
9535   \@@_msg_new:nn { Impossible-delimiter }
9536   {
9537     Impossible-delimiter.\
9538     It's-impossible-to-draw~the~#1~delimiter~of~your~
9539     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9540     in~that~column.
9541     \bool_if:NT \l_@@_submatrix_slim_bool
9542     { ~Maybe-you-should-try-without-the-key-'slim'. } \
9543     This~\token_to_str:N \SubMatrix\ will-be-ignored.
9544   }
9545   \@@_msg_new:nnn { width-without-X-columns }
9546   {
9547     You-have-used-the-key~'width'~but~you-have-put~no~'X'~column.~
9548     That-key-will-be-ignored.
9549   }
9550   {
9551     This-message-is~the-message-'width-without-X-columns'~

```

```

9552   of~the~module~'nicematrix'.~
9553   The~experimented~users~can~disable~that~message~with~
9554   \token_to_str:N \msg_redirect_name:nnn.\\
9555 }
9556
9557 \@@_msg_new:nn { key-multiplicity-with-dotted }
9558 {
9559   Incompatible-keys. \\
9560   You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9561   in~a~'custom-line'.~They~are~incompatible. \\
9562   The~key~'multiplicity'~will~be~discarded.
9563 }
9564 \@@_msg_new:nn { empty-environment }
9565 {
9566   Empty~environment.\\
9567   Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
9568 }
9569 \@@_msg_new:nn { No~letter~and~no~command }
9570 {
9571   Erroneous~use.\\
9572   Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
9573   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9574   '~'command'~(to~draw~horizontal~rules).\\
9575   However,~you~can~go~on.
9576 }
9577 \@@_msg_new:nn { Forbidden~letter }
9578 {
9579   Forbidden~letter.\\
9580   You~can't~use~the~letter~'#1'~for~a~customized~line.\\
9581   It~will~be~ignored.
9582 }
9583 \@@_msg_new:nn { Several~letters }
9584 {
9585   Wrong~name.\\
9586   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9587   have~used~'\l_@@_letter_str').\\
9588   It~will~be~ignored.
9589 }
9590 \@@_msg_new:nn { Delimiter~with~small }
9591 {
9592   Delimiter~forbidden.\\
9593   You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
9594   because~the~key~'small'~is~in~force.\\
9595   This~error~is~fatal.
9596 }
9597 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9598 {
9599   Unknown~cell.\\
9600   Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9601   the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9602   can't~be~executed~because~a~cell~doesn't~exist.\\
9603   This~command~\token_to_str:N \line\ will~be~ignored.
9604 }
9605 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
9606 {
9607   Duplicate~name.\\
9608   The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
9609   in~this~\@@_full_name_env:.\\
9610   This~key~will~be~ignored.\\
9611   \bool_if:NF \g_@@_messages_for_Overleaf_bool

```



```

9612     { For-a-list-of-the-names-already-used,-type-H<return>. }
9613 }
9614 {
9615   The-names-already-defined-in-this-\@@_full_name_env:\ are:~
9616   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9617 }
9618 \@@_msg_new:nn { r~or~l~with~preamble }
9619 {
9620   Erroneous-use.\\
9621   You-can't-use-the-key~'\l_keys_key_str'~in-your~\@@_full_name_env:~.~
9622   You-must-specify-the-alignment-of-your-columns-with-the-preamble-of~
9623   your~\@@_full_name_env:~.\\
9624   This-key-will-be-ignored.
9625 }
9626 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9627 {
9628   Erroneous-use.\\
9629   You-can't-use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9630   the~array.~This~error~is~fatal.
9631 }
9632 \@@_msg_new:nn { bad~corner }
9633 {
9634   Bad-corner.\\
9635   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9636   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
9637   This~specification~of~corner~will~be~ignored.
9638 }
9639 \@@_msg_new:nn { bad~border }
9640 {
9641   Bad-border.\\
9642   \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9643   (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9644   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9645   also~use~the~key~'tikz'
9646   \IfPackageLoadedF { tikz }
9647   {-if~you~load~the~LaTeX~package~'tikz'}).\\
9648   This~specification~of~border~will~be~ignored.
9649 }
9650 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9651 {
9652   TikZ~not~loaded.\\
9653   You-can't-use~\token_to_str:N \TikzEveryCell\
9654   because~you~have~not~loaded~tikz.~
9655   This~command~will~be~ignored.
9656 }
9657 \@@_msg_new:nn { tikz~key~without~tikz }
9658 {
9659   TikZ~not~loaded.\\
9660   You-can't-use~the~key~'tikz'~for~the~command~'\token_to_str:N
9661   \Block'~because~you~have~not~loaded~tikz.~
9662   This~key~will~be~ignored.
9663 }
9664 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
9665 {
9666   Erroneous-use.\\
9667   In~the~\@@_full_name_env:,~you~must~use~the~key~
9668   'last~col'~without~value.\\
9669   However,~you~can~go~on~for~this~time~
9670   (the~value~'\l_keys_value_tl'~will~be~ignored).
9671 }

```

```

9672 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
9673 {
9674   Erroneous-use.\
9675   In~\token_to_str:N \NiceMatrixOptions,~you-must-use-the-key~
9676   'last-col'~without-value.\
9677   However,~you-can-go-on-for-this-time~
9678   (the-value~'\l_keys_value_tl'~will-be-ignored).
9679 }

9680 \@@_msg_new:nn { Block-too-large-1 }
9681 {
9682   Block-too-large.\
9683   You-try-to-draw-a-block-in-the-cell-#1-#2-of-your-matrix-but-the-matrix-is~
9684   too-small-for-that-block. \
9685   This-block-and-maybe-others-will-be-ignored.
9686 }

9687 \@@_msg_new:nn { Block-too-large-2 }
9688 {
9689   Block-too-large.\
9690   The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
9691   \g_@@_static_num_of_col_int\
9692   columns-but-you-use-only~\int_use:N \c@jCol\ and-that's-why-a-block~
9693   specified-in-the-cell-#1-#2-can't-be-drawn.~You-should-add-some-ampersands~
9694   (&)~at-the-end-of-the-first-row-of-your~\@@_full_name_env:.\
9695   This-block-and-maybe-others-will-be-ignored.
9696 }

9697 \@@_msg_new:nn { unknown-column-type }
9698 {
9699   Bad-column-type.\
9700   The-column-type~'#1'~in-your~\@@_full_name_env:\
9701   is-unknown. \
9702   This-error-is-fatal.
9703 }

9704 \@@_msg_new:nn { unknown-column-type-S }
9705 {
9706   Bad-column-type.\
9707   The-column-type~'S'~in-your~\@@_full_name_env:\ is-unknown. \
9708   If-you-want-to-use-the-column-type~'S'~of~siunitx,~you-should~
9709   load-that-package. \
9710   This-error-is-fatal.
9711 }

9712 \@@_msg_new:nn { tabularnote-forbidden }
9713 {
9714   Forbidden-command.\
9715   You-can't-use-the-command~\token_to_str:N\tabularnote\
9716   ~here.~This-command-is-available-only-in~
9717   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9718   the-argument-of-a-command~\token_to_str:N \caption\ included~
9719   in-an-environment~{table}. \
9720   This-command-will-be-ignored.
9721 }

9722 \@@_msg_new:nn { borders-forbidden }
9723 {
9724   Forbidden-key.\
9725   You-can't-use-the-key~'borders'~of~the-command~\token_to_str:N \Block\
9726   because-the-option~'rounded-corners'~
9727   is-in-force-with-a-non-zero-value.\
9728   This-key-will-be-ignored.
9729 }

9730 \@@_msg_new:nn { bottomrule-without-booktabs }
9731 {
9732   booktabs-not-loaded.\

```

```

9733     You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9734     loaded~'booktabs'.\\
9735     This~key~will~be~ignored.
9736   }

9737   \@@_msg_new:nn { enumitem~not~loaded }
9738   {
9739     enumitem~not~loaded.\\
9740     You~can't~use~the~command~\token_to_str:N\tabularnote\
9741     ~because~you~haven't~loaded~'enumitem'.\\
9742     All~the~commands~\token_to_str:N\tabularnote\ will~be~
9743     ignored~in~the~document.
9744   }

9745   \@@_msg_new:nn { tikz~without~tikz }
9746   {
9747     Tikz~not~loaded.\\
9748     You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9749     loaded.~If~you~go~on,~that~key~will~be~ignored.
9750   }

9751   \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9752   {
9753     Tikz~not~loaded.\\
9754     You~have~used~the~key~'tikz'~in~the~definition~of~a~
9755     customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9756     You~can~go~on~but~you~will~have~another~error~if~you~actually~
9757     use~that~custom~line.
9758   }

9759   \@@_msg_new:nn { tikz~in~borders~without~tikz }
9760   {
9761     Tikz~not~loaded.\\
9762     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9763     command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9764     That~key~will~be~ignored.
9765   }

9766   \@@_msg_new:nn { without~color~inside }
9767   {
9768     If~order~to~use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9769     \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9770     outside~\token_to_str:N \CodeBefore,~you~
9771     should~have~used~the~key~'color~inside'~in~your~\@@_full_name_env:.\\
9772     You~can~go~on~but~you~may~need~more~compilations.
9773   }

9774   \@@_msg_new:nn { color~in~custom~line~with~tikz }
9775   {
9776     Erroneous~use.\\
9777     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
9778     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9779     The~key~'color'~will~be~discarded.
9780   }

9781   \@@_msg_new:nn { Wrong~last~row }
9782   {
9783     Wrong~number.\\
9784     You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
9785     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9786     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9787     last~row.~You~can~avoid~this~problem~by~using~'last~row'~
9788     without~value~(more~compilations~might~be~necessary).
9789   }

9790   \@@_msg_new:nn { Yet~in~env }
9791   {
9792     Nested~environments.\\

```

```

9793     Environments-of-nicematrix-can't-be-nested.\
9794     This-error-is-fatal.
9795 }
9796 \@@_msg_new:nn { Outside-math-mode }
9797 {
9798     Outside-math-mode.\
9799     The-\@@_full_name_env:\ can-be-used-only-in-math-mode~
9800     (and-not-in-\token_to_str:N \vcenter).\
9801     This-error-is-fatal.
9802 }
9803 \@@_msg_new:nn { One-letter-allowed }
9804 {
9805     Bad-name.\
9806     The-value-of-key~'\l_keys_key_str'~must-be-of-length-1.\
9807     It-will-be-ignored.
9808 }
9809 \@@_msg_new:nn { TabularNote-in-CodeAfter }
9810 {
9811     Environment~{TabularNote}~forbidden.\
9812     You-must-use~{TabularNote}~at-the-end-of-your~{NiceTabular}~
9813     but~*before*~the~\token_to_str:N \CodeAfter.\
9814     This-environment~{TabularNote}~will-be-ignored.
9815 }
9816 \@@_msg_new:nn { varwidth-not-loaded }
9817 {
9818     varwidth-not-loaded.\
9819     You-can't-use-the-column-type~'V'~because~'varwidth'~is-not~
9820     loaded.\
9821     Your-column-will-behave-like~'p'.
9822 }
9823 \@@_msg_new:nnn { Unknow-key-for-RulesBis }
9824 {
9825     Unknow-key.\
9826     Your-key~'\l_keys_key_str'~is-unknown-for-a-rule.\
9827     \c_@@_available_keys_str
9828 }
9829 {
9830     The-available-keys-are-(in-alphabetic-order):~
9831     color,~
9832     dotted,~
9833     multiplicity,~
9834     sep-color,~
9835     tikz,~and~total-width.
9836 }
9837
9838 \@@_msg_new:nnn { Unknow-key-for-Block }
9839 {
9840     Unknow-key.\
9841     The-key~'\l_keys_key_str'~is-unknown-for-the-command~\token_to_str:N
9842     \Block.\ It-will-be-ignored. \
9843     \c_@@_available_keys_str
9844 }
9845 {
9846     The-available-keys-are-(in-alphabetic-order):~&-in-blocks,~ampersand-in-blocks,~
9847     b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
9848     opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
9849     and~vlines.
9850 }
9851 \@@_msg_new:nnn { Unknow-key-for-Brace }
9852 {
9853     Unknow-key.\

```

```

9854 The-key~'\l_keys_key_str'~is-unknown~for~the~commands~\token_to_str:N
9855 \UnderBrace\ and~\token_to_str:N \OverBrace.\
9856 It~will~be~ignored. \
9857 \c_@@_available_keys_str
9858 }
9859 {
9860 The-available-keys-are~(in~alphabetic~order):~color,~left-shorten,~
9861 right-shorten,~shorten~(which~fixes~both~left~shorten~and~
9862 right-shorten)~and~yshift.
9863 }
9864 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9865 {
9866 Unknown~key.\
9867 The-key~'\l_keys_key_str'~is-unknown.\
9868 It~will~be~ignored. \
9869 \c_@@_available_keys_str
9870 }
9871 {
9872 The-available-keys-are~(in~alphabetic~order):~
9873 delimiters/color,~
9874 rules~(with~the~subkeys~'color'~and~'width'),~
9875 sub-matrix~(several~subkeys)~
9876 and~xdots~(several~subkeys).~
9877 The-latter~is~for~the~command~\token_to_str:N \line.
9878 }
9879 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9880 {
9881 Unknown~key.\
9882 The-key~'\l_keys_key_str'~is-unknown.\
9883 It~will~be~ignored. \
9884 \c_@@_available_keys_str
9885 }
9886 {
9887 The-available-keys-are~(in~alphabetic~order):~
9888 create-cell-nodes,~
9889 delimiters/color~and~
9890 sub-matrix~(several~subkeys).
9891 }
9892 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9893 {
9894 Unknown~key.\
9895 The-key~'\l_keys_key_str'~is-unknown.\
9896 That~key~will~be~ignored. \
9897 \c_@@_available_keys_str
9898 }
9899 {
9900 The-available-keys-are~(in~alphabetic~order):~
9901 'delimiters/color',~
9902 'extra-height',~
9903 'hlines',~
9904 'hvlines',~
9905 'left-xshift',~
9906 'name',~
9907 'right-xshift',~
9908 'rules'~(with~the~subkeys~'color'~and~'width'),~
9909 'slim',~
9910 'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9911 and~'right-xshift').\
9912 }
9913 \@@_msg_new:nnn { Unknown~key~for~notes }
9914 {
9915 Unknown~key.\

```

```

9916     The-key~'\l_keys_key_str'~is-unknown.\\
9917     That-key-will-be-ignored. \\
9918     \c_@@_available_keys_str
9919 }
9920 {
9921     The-available-keys-are-(in~alphabetic-order):~
9922     bottomrule,~
9923     code-after,~
9924     code-before,~
9925     detect-duplicates,~
9926     enumitem-keys,~
9927     enumitem-keys-para,~
9928     para,~
9929     label-in-list,~
9930     label-in-tabular~and~
9931     style.
9932 }
9933 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9934 {
9935     Unknown~key.\\
9936     The-key~'\l_keys_key_str'~is-unknown~for~the~command~
9937     \token_to_str:N \RowStyle. \\
9938     That-key-will-be-ignored. \\
9939     \c_@@_available_keys_str
9940 }
9941 {
9942     The-available-keys-are-(in~alphabetic-order):~
9943     'bold',~
9944     'cell-space-top-limit',~
9945     'cell-space-bottom-limit',~
9946     'cell-space-limits',~
9947     'color',~
9948     'nb-rows'~and~
9949     'rowcolor'.
9950 }
9951 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9952 {
9953     Unknown~key.\\
9954     The-key~'\l_keys_key_str'~is-unknown~for~the~command~
9955     \token_to_str:N \NiceMatrixOptions. \\
9956     That-key-will-be-ignored. \\
9957     \c_@@_available_keys_str
9958 }
9959 {
9960     The-available-keys-are-(in~alphabetic-order):~
9961     &in-blocks,~
9962     allow-duplicate-names,~
9963     ampersand-in-blocks,~
9964     caption-above,~
9965     cell-space-bottom-limit,~
9966     cell-space-limits,~
9967     cell-space-top-limit,~
9968     code-for-first-col,~
9969     code-for-first-row,~
9970     code-for-last-col,~
9971     code-for-last-row,~
9972     corners,~
9973     custom-key,~
9974     create-extra-nodes,~
9975     create-medium-nodes,~
9976     create-large-nodes,~
9977     custom-line,~
9978     delimiters~(several~subkeys),~

```

```

9979     end-of-row,~
9980     first-col,~
9981     first-row,~
9982     hlines,~
9983     hvlines,~
9984     hvlines-except-borders,~
9985     last-col,~
9986     last-row,~
9987     left-margin,~
9988     light-syntax,~
9989     light-syntax-expanded,~
9990     matrix/columns-type,~
9991     no-cell-nodes,~
9992     notes~(several~subkeys),~
9993     nullify-dots,~
9994     pgf-node-code,~
9995     renew-dots,~
9996     renew-matrix,~
9997     respect-arraystretch,~
9998     rounded-corners,~
9999     right-margin,~
10000     rules~(with~the~subkeys~'color'~and~'width'),~
10001     small,~
10002     sub-matrix~(several~subkeys),~
10003     vlines,~
10004     xdots~(several~subkeys).
10005 }

```

For `{NiceArray}`, the set of keys is the same as for `{NiceMatrix}` excepted that there is no `l` and `r`.

```

10006 \@@_msg_new:nnn { Unknown-key-for-NiceArray }
10007 {
10008   Unknown-key.\\
10009   The-key~'\l_keys_key_str'~is~unknown~for~the~environment~
10010   \{NiceArray\}. \\
10011   That~key~will~be~ignored. \\
10012   \c_@@_available_keys_str
10013 }
10014 {
10015   The-available-keys~are~(in~alphabetic~order):~
10016   &~in~blocks,~
10017   ampersand-in-blocks,~
10018   b,~
10019   baseline,~
10020   c,~
10021   cell-space-bottom-limit,~
10022   cell-space-limits,~
10023   cell-space-top-limit,~
10024   code-after,~
10025   code-for-first-col,~
10026   code-for-first-row,~
10027   code-for-last-col,~
10028   code-for-last-row,~
10029   color-inside,~
10030   columns-width,~
10031   corners,~
10032   create-extra-nodes,~
10033   create-medium-nodes,~
10034   create-large-nodes,~
10035   extra-left-margin,~
10036   extra-right-margin,~
10037   first-col,~
10038   first-row,~
10039   hlines,~

```

```

10040   hvlines,~
10041   hvlines-except-borders,~
10042   last-col,~
10043   last-row,~
10044   left-margin,~
10045   light-syntax,~
10046   light-syntax-expanded,~
10047   name,~
10048   no-cell-nodes,~
10049   nullify-dots,~
10050   pgf-node-code,~
10051   renew-dots,~
10052   respect-arraystretch,~
10053   right-margin,~
10054   rounded-corners,~
10055   rules~(with~the~subkeys~'color'~and~'width'),~
10056   small,~
10057   t,~
10058   vlines,~
10059   xdots/color,~
10060   xdots/shorten-start,~
10061   xdots/shorten-end,~
10062   xdots/shorten-and~
10063   xdots/line-style.
10064 }

```

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`).

```

10065 \@@_msg_new:nmn { Unknown~key~for~NiceMatrix }
10066 {
10067   Unknown~key.\\
10068   The~key~'\l_keys_key_str'~is~unknown~for~the~
10069   \@@_full_name_env:. \\
10070   That~key~will~be~ignored. \\
10071   \c_@@_available_keys_str
10072 }
10073 {
10074   The~available~keys~are~(in~alphabetic~order):~
10075   &~in~blocks,~
10076   ampersand~in~blocks,~
10077   b,~
10078   baseline,~
10079   c,~
10080   cell-space-bottom-limit,~
10081   cell-space-limits,~
10082   cell-space-top-limit,~
10083   code-after,~
10084   code-for-first-col,~
10085   code-for-first-row,~
10086   code-for-last-col,~
10087   code-for-last-row,~
10088   color-inside,~
10089   columns-type,~
10090   columns-width,~
10091   corners,~
10092   create-extra-nodes,~
10093   create-medium-nodes,~
10094   create-large-nodes,~
10095   extra-left-margin,~
10096   extra-right-margin,~
10097   first-col,~
10098   first-row,~
10099   hlines,~
10100   hvlines,~

```



```

10101   hvlines-except-borders,~
10102   l,~
10103   last-col,~
10104   last-row,~
10105   left-margin,~
10106   light-syntax,~
10107   light-syntax-expanded,~
10108   name,~
10109   no-cell-nodes,~
10110   nullify-dots,~
10111   pgf-node-code,~
10112   r,~
10113   renew-dots,~
10114   respect-arraystretch,~
10115   right-margin,~
10116   rounded-corners,~
10117   rules~(with~the~subkeys~'color'~and~'width'),~
10118   small,~
10119   t,~
10120   vlines,~
10121   xdots/color,~
10122   xdots/shorten-start,~
10123   xdots/shorten-end,~
10124   xdots/shorten-and~
10125   xdots/line-style.
10126 }
10127 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10128 {
10129   Unknown~key.\\
10130   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10131   \{NiceTabular\}. \\
10132   That~key~will~be~ignored. \\
10133   \c_@@_available_keys_str
10134 }
10135 {
10136   The~available~keys~are~(in~alphabetic~order):~
10137   &~in~blocks,~
10138   ampersand~in~blocks,~
10139   b,~
10140   baseline,~
10141   c,~
10142   caption,~
10143   cell-space-bottom-limit,~
10144   cell-space-limits,~
10145   cell-space-top-limit,~
10146   code~after,~
10147   code~for~first~col,~
10148   code~for~first~row,~
10149   code~for~last~col,~
10150   code~for~last~row,~
10151   color~inside,~
10152   columns~width,~
10153   corners,~
10154   custom~line,~
10155   create~extra~nodes,~
10156   create~medium~nodes,~
10157   create~large~nodes,~
10158   extra~left~margin,~
10159   extra~right~margin,~
10160   first~col,~
10161   first~row,~
10162   hlines,~
10163   hvlines,~

```

```

10164   hvlines-except-borders,~
10165   label,~
10166   last-col,~
10167   last-row,~
10168   left-margin,~
10169   light-syntax,~
10170   light-syntax-expanded,~
10171   name,~
10172   no-cell-nodes,~
10173   notes~(several~subkeys),~
10174   nullify-dots,~
10175   pgf-node-code,~
10176   renew-dots,~
10177   respect-arraystretch,~
10178   right-margin,~
10179   rounded-corners,~
10180   rules~(with~the~subkeys~'color'~and~'width'),~
10181   short-caption,~
10182   t,~
10183   tabularnote,~
10184   vlines,~
10185   xdots/color,~
10186   xdots/shorten-start,~
10187   xdots/shorten-end,~
10188   xdots/shorten-and~
10189   xdots/line-style.
10190 }

10191 \@@_msg_new:nnn { Duplicate-name }
10192 {
10193   Duplicate~name.\\
10194   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
10195   the~same~environment~name~twice.~You~can~go~on,~but,~
10196   maybe,~you~will~have~incorrect~results~especially~
10197   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10198   message~again,~use~the~key~'allow-duplicate-names'~in~
10199   '\token_to_str:N \NiceMatrixOptions'.\\
10200   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10201     { For~a~list~of~the~names~already~used,~type-H~<return>. }
10202 }
10203 {
10204   The~names~already~defined~in~this~document~are:~
10205   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10206 }

10207 \@@_msg_new:nn { Option-auto-for-columns-width }
10208 {
10209   Erroneous~use.\\
10210   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
10211   That~key~will~be~ignored.
10212 }

10213 \@@_msg_new:nn { NiceTabularX~without~X }
10214 {
10215   NiceTabularX~without~X.\\
10216   You~should~not~use~{NiceTabularX}~without~X~columns.\\
10217   However,~you~can~go~on.
10218 }

10219 \@@_msg_new:nn { Preamble~forgotten }
10220 {
10221   Preamble~forgotten.\\
10222   You~have~probably~forgotten~the~preamble~of~your~
10223   \@@_full_name_env:. \\
10224   This~error~is~fatal.
10225 }

```

# Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	4
4	Parameters	8
5	The command <code>\tabularnote</code>	18
6	Command for creation of rectangle nodes	23
7	The options	24
8	Important code used by <code>{NiceArrayWithDelims}</code>	35
9	The <code>\CodeBefore</code>	49
10	The environment <code>{NiceArrayWithDelims}</code>	53
11	We construct the preamble of the array	58
12	The redefinition of <code>\multicolumn</code>	73
13	The environment <code>{NiceMatrix}</code> and its variants	91
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	92
15	After the construction of the array	93
16	We draw the dotted lines	100
17	The actual instructions for drawing the dotted lines with <code>Tikz</code>	113
18	User commands available in the new environments	119
19	The command <code>\line</code> accessible in code-after	125
20	The command <code>\RowStyle</code>	127
21	Colors of cells, rows and columns	129
22	The vertical and horizontal rules	141
23	The empty corners	156
24	The environment <code>{NiceMatrixBlock}</code>	158
25	The extra nodes	160
26	The blocks	164
27	How to draw the dotted lines transparently	188
28	Automatic arrays	188
29	The redefinition of the command <code>\dotfill</code>	190
30	The command <code>\diagbox</code>	190

<b>31</b>	<b>The keyword <code>\CodeAfter</code></b>	<b>191</b>
<b>32</b>	<b>The delimiters in the preamble</b>	<b>192</b>
<b>33</b>	<b>The command <code>\SubMatrix</code></b>	<b>193</b>
<b>34</b>	<b>Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code></b>	<b>202</b>
<b>35</b>	<b>The command <code>TikzEveryCell</code></b>	<b>205</b>
<b>36</b>	<b>The command <code>\ShowCellNames</code></b>	<b>206</b>
<b>37</b>	<b>We process the options at package loading</b>	<b>208</b>
<b>38</b>	<b>About the package underscore</b>	<b>209</b>
<b>39</b>	<b>Error messages of the package</b>	<b>210</b>